การโปรแกรมคอมพิวเตอร์
Computer Programming



สาขาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์และเทคโนโลยี

FET
Faculty of Engineering and Technology
Rajamangala University of Technology Isan
WWW.FET.RMUTI.AC.TH

พิมพ์ ครั้งที่ 1

คำนำ

วิชาการโปรแกรมคอมพิวเตอร์เป็นวิชาพื้นฐานที่สำคัญ ซึ่งทุกสาขาวิชาต้องเรียนอันเนื่องจากทุกสายงานด้านต่างๆ มีการประยุกต์ใช้งานคอมพิวเตอร์ สืบเนื่องจากจุดเด่นของคอมพิวเตอร์ สามารถทำงานซ้ำ ๆ สามารถคำนวณที่รวดเร็ว และ แม่นยำ สามารถจัดเก็บข้อมูลได้ และสามารถสร้างโปรแกรมประยุกต์ใช้งานได้ในทุกสาขาวิชาชีพ โดยเฉพาะอย่างยิ่งในกลุ่มที่ศึกษาด้านคอมพิวเตอร์ และวิศวกรรมที่จำเป็นต้องใช้งานคอมพิวเตอร์ การศึกษาการโปรแกรมคอมพิวเตอร์เป็นทั้งศาสตร์ และศิลป์ ที่จะต้องมีการฝึกฝน เรียนรู้เทคนิคต่าง ๆ และต้องใช้งานบ่อย ๆ จึงจะเกิดความสำเร็จได้ หนังสือนี้คาดหวังว่าจะเกิดประโยชน์ และสามารถสร้างแนวคิดใหม่ ๆ สำหรับผู้เรียนและผู้สนใจ ในการโปรแกรมคอมพิวเตอร์

หนังสือการโปรแกรมคอมพิวเตอร์นี้ ประกอบด้วย 12 บทเรียน เริ่มจากหลักการเบื้องต้นขององค์ประกอบระบบคอมพิวเตอร์ การปฏิสัมพันธ์ระหว่างฮาร์ดแวร์และซอฟต์แวร์ และ การประมวลผลอิเล็กทรอนิกส์ การพัฒนาโปรแกรม ผังงาน โครงสร้างข้อมูลและตัวแปร การดำเนินงานทางคณิตศาสตร์และตรรกศาสตร์ การรับข้อมูลและการส่งออก การติดต่อกับผู้ใช้ การเขียนโปรแกรมโครงสร้าง คำสั่งตัดสินใจ การเขียนโปรแกรมแบบวนรอบ การเขียนฟังก์ชัน การจัดการข้อมูลแถวลำดับ และ ข้อความ การสร้างข้อมูลชนิดโครงสร้าง การดำเนินงานแฟ้มข้อมูล ดังนี้ หากเข้าใจและสนุกกับการเขียนโปรแกรมก็สามารถพัฒนาตัวเองได้ เป็นการพัฒนาบุคคล พัฒนาองค์กร และพัฒนาชาติได้

ทีมงานวิศวกรรมคอมพิวเตอร์

พฤษภาคม 2565

สารบัญ

หน้า

สารบัญ

สารบัญ

หน้า

สารบัญ

บทที่ 1
หลักการเบื้องต้นขององค์ประกอบระบบคอมพิวเตอร์

1.1 บทนำ

คอมพิวเตอร์ คือ อุปกรณ์อิเล็กทรอนิกส์ ทำงานตามชุดคำสั่งอัตโนมัติ ที่สามารถรับข้อมูลและคำสั่งผ่านอุปกรณ์รับข้อมูล แล้วนำข้อมูลและคำสั่งนั้นไปประมวลผลด้วยหน่วยประมวลผล เพื่อให้ได้ผลลัพธ์ที่ต้องการและแสดงผลผ่านอุปกรณ์แสดงผล ตลอดจนสามารถบันทึกรายการต่างๆไว้เพื่อใช้งานได้ด้วยอุปกรณ์บันทึกข้อมูลสำรอง

1.2 คุณสมบัติของคอมพิวเตอร์

ในปัจจุบันผู้ใช้ส่วนใหญ่มักจะคิดว่าคอมพิวเตอร์ เป็นเครื่องมือที่ทำงานได้สารพัด แต่ผู้ที่มีความรู้ทางคอมพิวเตอร์จะทราบว่างานที่เหมาะสมกับการนำคอมพิวเตอร์มาใช้อย่างยิ่งคือ การสร้างสารสนเทศ ซึ่งสารสนเทศเหล่านั้นสามารถนำมาพิมพ์ออกทางเครื่องพิมพ์ ส่งผ่านเครือข่ายคอมพิวเตอร์หรือจัดเก็บไว้ใช้ในอนาคตก็ได้ เนื่องจากคอมพิวเตอร์จะมีคุณสมบัติต่าง ๆ ดังต่อไปนี้

1.2.1 ความเป็นอัตโนมัติ (Self acting) การทำงานของคอมพิวเตอร์จะทำงานแบบอัตโนมัติภายใต้คำสั่งที่ได้กำหนดไว้ การทำงานดังกล่าวจะเริ่มตั้งแต่การนำข้อมูลเข้าสู่ระบบการประมวลผลและแปลงผลลัพธ์ออกมาให้อยู่ในรูปแบบที่มนุษย์เข้าใจได้

1.2.2 ความเร็ว (Speed) คอมพิวเตอร์ในปัจจุบันนี้ สามารถทำงานได้ถึงร้อยล้านคำสั่งในหนึ่งวินาที

1.2.3 ความเชื่อถือ (Reliable) คอมพิวเตอร์ทุกวันนี้จะทำงานได้ทั้งกลางวันและกลางคืนอย่างไม่มีข้อผิดพลาดและไม่รู้จักเหน็ดเหนื่อย

1.2.4 ความถูกต้องแม่นยำ (Accurate) วงจรคอมพิวเตอร์นั้นจะให้ผลของการคำนวณที่ถูกต้องเสมอหากผลการคำนวณผิดจากที่ควรจะเป็น มักเกิดจากความผิดพลาดของโปรแกรม หรือข้อมูลที่เข้าสู่โปรแกรม

1.2.5 เก็บข้อมูลจำนวนมาก (Store massive amounts of information) ไมโครคอมพิวเตอร์ในปัจจุบันจะมีที่เก็บข้อมูลสำรองที่มีความสูงมากกว่าหนึ่งพันล้านตัวอักษร และสำหรับคอมพิวเตอร์ขนาดใหญ่จะสามารถเก็บข้อมูลได้มากกว่าหนึ่งล้านๆ ตัวอักษร

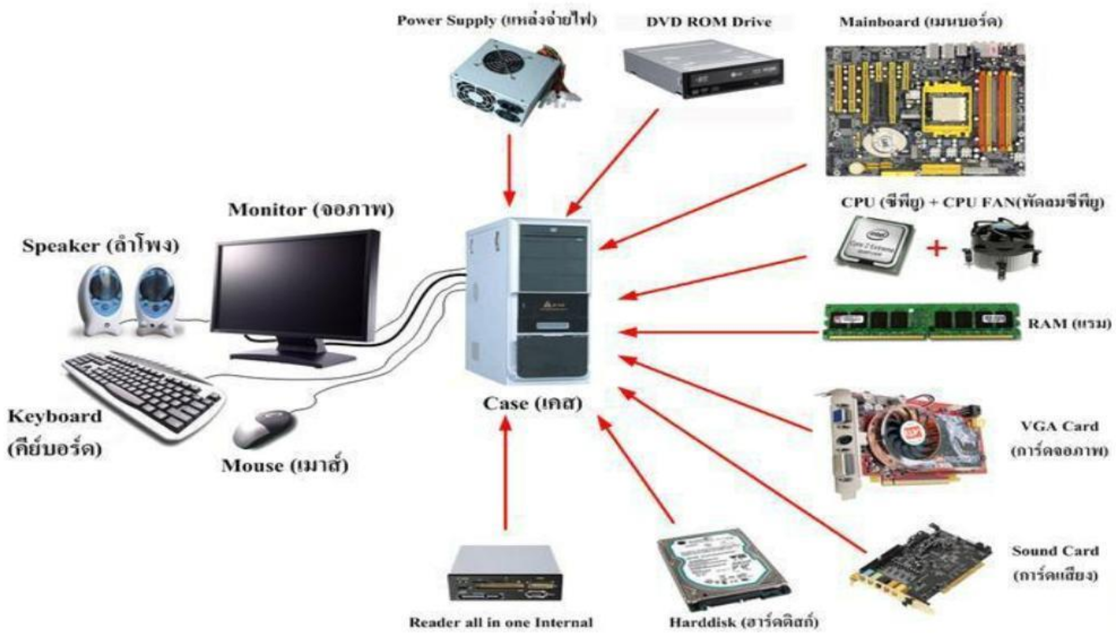
1.2.6 ย้ายข้อมูลจากที่หนึ่งไปยังอีกที่หนึ่งได้อย่างรวดเร็ว (Move information) โดยใช้การติดต่อสื่อสารผ่านระบบเครือข่ายคอมพิวเตอร์ ซึ่งสามารถส่งพจนานุกรมหนึ่งเล่ม ในรูปของข้อมูลอิเล็กทรอนิกส์ ไปยังเครื่องคอมพิวเตอร์ที่อยู่ไกลคนละซีกโลกได้ในเวลาเพียงไม่ถึงหนึ่งนาที ทำให้มีการเรียกเครือข่ายคอมพิวเตอร์ที่เชื่อมกัน ทั่วโลกในปัจจุบันว่าทางด่วนสารสนเทศ (Information Superhighway)

1.2.7 ทำงานซ้ำ ๆ ได้ (Repeatability) ช่วยลดปัญหาเรื่องความอ่อนล้าจากการทำงานของแรงงานคน นอกจากนี้ยังลดความผิดพลาดต่าง ๆ ได้ดีกว่าด้วยข้อมูลที่ประมวลผลแม้จะยุ่งยากหรือซับซ้อนเพียงใดก็ตามจะสามารถคำนวณและหาผลลัพธ์ได้อย่างรวดเร็ว

1.3 องค์ประกอบของคอมพิวเตอร์

คอมพิวเตอร์ทำงานได้อย่างเป็นระบบและมีประสิทธิภาพแล้วระบบคอมพิวเตอร์ควรจะประกอบไปด้วยองค์ประกอบสำคัญ 4 ส่วนประกอบไปด้วย ฮาร์ดแวร์ (Hardware) ซอฟต์แวร์ (Software) บุคลากร (People ware) ข้อมูล / สารสนเทศ (Data/Information)

1.3.1 ฮาร์ดแวร์ หมายถึง อุปกรณ์ต่างๆ ที่ประกอบขึ้นเป็นเครื่องคอมพิวเตอร์ มีลักษณะเป็น โครงร่างสามารถมองเห็นด้วยตาและสัมผัสได้ เช่น จอภาพ คีย์บอร์ด เครื่องพิมพ์ เมาส์ เป็นต้น ซึ่งสามารถแบ่งออกเป็นส่วนต่างๆ ตามลักษณะการทำงาน ได้ 4 หน่วย คือ หน่วยรับข้อมูล (Input Unit) หน่วยประมวลผลกลาง (Central Processing Unit : CPU) หน่วยแสดงผล (Output Unit) หน่วยเก็บข้อมูลสำรอง (Secondary Storage) โดยอุปกรณ์แต่ละหน่วยมีหน้าที่การทำงานแตกต่างกัน



รูปที่ 1.1 ส่วนประกอบของฮาร์ดแวร์

1.3.1.1 อุปกรณ์นำข้อมูลเข้า (Input Device) เป็นอุปกรณ์ที่เกี่ยวข้องกับการนำเข้าข้อมูลหรือชุดคำสั่งเข้ามายังระบบ เพื่อให้คอมพิวเตอร์ประมวลผลต่อไปได้ซึ่งอาจจะเป็น ตัวเลข ภาพนิ่ง ภาพเคลื่อนไหว เสียง ได้แก่ คีย์บอร์ด (Keyboard) เมาส์ (Mouse) สแกนเนอร์ (Scanner) จอสัมผัส (Touch screen) เว็บแคม (Webcam) ไมโครโฟน (Microphone) เป็นต้น

1.3.1.2 อุปกรณ์ประมวลผล (Processing Device) หน่วยประมวลผลกลางนิยมเรียกสั้น ๆ ว่า ซีพียู (CPU : Central Processing Unit) เรียกอีกชื่อหนึ่งว่า โปรเซสเซอร์ (Processor) หรือ ชิป (Chip) ทำหน้าที่ในการทำงานตามคำสั่งที่ปรากฏอยู่ในโปรแกรม ปัจจุบันซีพียูของเครื่องพีซี หรือ Chip ตัวอย่าง เช่น ของบริษัท Intel คือ Core i5 ส่วนของบริษัท AMD คือ Ryzen ซึ่งหน่วยประมวลผลกลาง ประกอบไปด้วยส่วนที่สำคัญ 3 ส่วน คือ

1) หน่วยคำนวณและตรรกะ (Arithmetic & Logical Unit : ALU) ทำหน้าที่เหมือนกับเครื่องคำนวณอยู่ในคอมพิวเตอร์ โดยทำงานเกี่ยวข้องกับการคำนวณทางคณิตศาสตร์ เช่น บวก ลบ คูณ หาร นอกจากนี้หน่วยคำนวณและตรรกะของคอมพิวเตอร์ ยังมีความสามารถอีกอย่างหนึ่ง ที่เครื่องคำนวณธรรมดาไม่มีคือความสามารถในเชิงตรรกะศาสตร์หมายถึงความ สามารถในการเปรียบเทียบตามเงื่อนไขและกฎเกณฑ์ทางคอมพิวเตอร์ เพื่อให้ได้คำตอบออกมาว่าเงื่อนไขนั้นเป็นจริง หรือเป็นเท็จ เช่น เปรียบเทียบมากกว่า น้อยกว่า เท่ากัน ไม่เท่ากัน ของจำนวน 2 จำนวนเป็นต้น ซึ่งการเปรียบเทียบนี้มักจะใช้ในการเลือกทำงานของเครื่องคอมพิวเตอร์ จะทำตามคำสั่งใดของโปรแกรมเป็นคำสั่งต่อไป

2) หน่วยควบคุม (Control Unit) ทำหน้าที่ ควบคุมลำดับขั้นตอนการประมวลผล และการทำงานของอุปกรณ์ต่าง ๆ ภายในหน่วยประมวลผลกลาง และรวมไปถึงการประสานงานในการทำงานร่วมกันระหว่างหน่วยประมวลผลกลาง กับอุปกรณ์นำเข้าข้อมูล อุปกรณ์แสดงผล และหน่วยความจำสำรองด้วย เมื่อผู้ใช้ต้องการประมวลผล ตามชุดคำสั่งใด ผู้ใช้จะต้องส่งข้อมูลและชุดคำสั่งนั้น ๆ เข้าสู่ระบบคอมพิวเตอร์เสียก่อน โดยข้อมูลและชุดคำสั่งดังกล่าวจะถูกนำไปเก็บไว้ในหน่วยความจำหลักก่อน จากนั้นหน่วยควบคุมจะดึงคำสั่งจากชุดคำสั่งที่มีอยู่ในหน่วยความจำหลักออกมาทีละคำสั่ง เพื่อทำการแปล ความหมายว่าคำสั่งดังกล่าวสั่งให้ฮาร์ดแวร์ส่วนใดทำงานอะไรกับข้อมูลตัวใด เมื่อทราบความหมายของคำสั่งนั้นแล้วหน่วยควบคุมก็จะส่งสัญญาณคำสั่งไปยังฮาร์ดแวร์ ส่วนที่ทำหน้าที่ในประมวลผลดังกล่าว ให้ทำตามคำสั่งนั้น ๆ เช่น ถ้าคำสั่งที่เข้ามานั้นเป็นคำสั่งเกี่ยวกับการคำนวณ หน่วยควบคุมจะส่งสัญญาณคำสั่งไปยังหน่วยคำนวณ และตรรกะให้ทำงาน หน่วยคำนวณและตรรกะ ก็จะไปทำการดึงข้อมูลจาก หน่วยความจำหลักเข้ามาประมวลผลตามคำสั่ง แล้วนำผลลัพธ์ที่ได้ไปแสดงยังอุปกรณ์แสดงผล หน่วยควบคุมจึงจะส่งสัญญาณคำสั่งไปยังอุปกรณ์แสดงผล ที่กำหนดให้ดึงข้อมูลจากหน่วยความจำหลัก ออกไปแสดงให้เห็นผลลัพธ์ดังกล่าวอีกต่อหนึ่ง

3) หน่วยความจำหลัก (Main Memory) คอมพิวเตอร์จะสามารถทำงานได้เมื่อมีข้อมูลและชุดคำสั่งที่ใช้ในการประมวลผลอยู่ใน หน่วยความจำหลักเรียบร้อยแล้วเท่านั้น และหลังจากการทำการ

ประมวลผลข้อมูลตามชุดคำสั่งเรียบร้อยแล้ว ผลลัพธ์ที่ได้จะถูกนำไปเก็บไว้ที่หน่วยความจำหลักและก่อนจะถูกนำออกไปแสดงที่อุปกรณ์แสดงผล

1.3.1.3 อุปกรณ์แสดงผล (Output device) คืออุปกรณ์สำหรับแสดงผลลัพธ์ที่ได้จากการประมวลผลของคอมพิวเตอร์และเป็นอุปกรณ์ส่งออก (Output device) ทำหน้าที่แสดง ผลลัพธ์เมื่อซีพียูทำการประมวลผล ได้แก่ จอภาพ (Monitor) เครื่องพิมพ์ (Printer) เครื่องฉายภาพข้ามศีรษะ (Overhead Projector) เป็นต้น

1.3.2 ซอฟต์แวร์ (Software) หมายถึง ส่วนที่มนุษย์สัมผัสไม่ได้โดยตรงเป็นโปรแกรมหรือชุดคำสั่งที่ถูกเขียนขึ้นเพื่อสั่งให้เครื่องคอมพิวเตอร์ทำงาน ซอฟต์แวร์จึงเป็นเหมือนตัวเชื่อมระหว่างผู้ใช้เครื่องคอมพิวเตอร์และเครื่องคอมพิวเตอร์ ถ้าไม่มีซอฟต์แวร์เราก็ไม่สามารถใช้เครื่องคอมพิวเตอร์ทำอะไรได้เลย ซอฟต์แวร์สำหรับเครื่องคอมพิวเตอร์สามารถแบ่งได้ดังนี้

1) ซอฟต์แวร์สำหรับระบบ (System Software) คือ ชุดของคำสั่งที่เขียนไว้เป็นคำสั่งสำเร็จรูป ซึ่งจะทำงานใกล้ชิดกับคอมพิวเตอร์มากที่สุด เพื่อคอยควบคุมการทำงานของฮาร์ดแวร์ทุกอย่าง และอำนวยความสะดวกให้กับผู้ใช้ในการใช้งาน ซอฟต์แวร์หรือโปรแกรมระบบที่รู้จักกันดีก็คือ DOS, Windows, UNIX, Linux รวมทั้งโปรแกรมแปลคำสั่งที่เขียนในภาษาระดับสูงเช่น ภาษา Basic, Python, C เป็นต้น

2) ซอฟต์แวร์ประยุกต์ (Application Software) คือ ซอฟต์แวร์หรือโปรแกรมที่สั่งคอมพิวเตอร์ทำงานต่างๆ ตามที่ผู้ใช้ต้องการ ไม่ว่าจะด้านเอกสาร บัญชีการจัดเก็บข้อมูล เป็นต้น ซอฟต์แวร์ประยุกต์สามารถจำแนกได้เป็น 2 ประเภท คือ

- ซอฟต์แวร์สำหรับงานเฉพาะด้าน คือ โปรแกรมซึ่งเขียนขึ้นเพื่อการทำงานเฉพาะอย่างที่เราต้องการบางที่เรียกว่า User's Program เช่นโปรแกรมการทำบัญชีจ่ายเงินเดือน โปรแกรมระบบเช่าซื้อ โปรแกรมการทำสินค้าคงคลัง เป็นต้น ซึ่งแต่ละโปรแกรมก็มักจะมีเงื่อนไขหรือแบบฟอร์มแตกต่างกันออกไปตามความต้องการ หรือกฎเกณฑ์ของแต่ละหน่วยงานที่ใช้ ซึ่งสามารถดัดแปลงแก้ไขเพิ่มเติม (Modifications) ในบางส่วนของโปรแกรมได้เพื่อให้ตรงกับความต้องการของผู้ใช้และซอฟต์แวร์ประยุกต์ที่เขียนขึ้นนี้โดยส่วนใหญ่มักใช้ภาษาระดับสูงเป็นตัวพัฒนา

- ซอฟต์แวร์สำหรับงานทั่วไป เป็นโปรแกรมประยุกต์ที่มีผู้จัดทำไว้ เพื่อใช้ในการทำงานประเภทต่างๆ ทั่วไป โดยผู้ใช้คนอื่นๆ สามารถนำโปรแกรมนี้ไปประยุกต์ใช้กับข้อมูลของตนได้แต่จะไม่สามารถทำการดัดแปลง หรือแก้ไขโปรแกรมได้ผู้ใช้ไม่จำเป็นต้องเขียนโปรแกรมเอง ซึ่งเป็นการประหยัดเวลา แรงงาน และค่าใช้จ่ายในการเขียนโปรแกรม นอกจากนี้ ยังไม่ต้องใช้เวลามากในการฝึกและปฏิบัติ ซึ่งโปรแกรมสำเร็จรูปนี้มักจะมีการใช้งานในหน่วยงานที่ขาดบุคลากรที่มีความชำนาญเป็นพิเศษในการเขียนโปรแกรม ดังนั้นการใช้โปรแกรมสำเร็จรูปจึงเป็นสิ่งที่อำนวยความสะดวกและเป็นประโยชน์อย่างยิ่ง ตัวอย่างโปรแกรมสำเร็จรูปที่นิยมใช้ได้แก่ MS - Office, Adobe Photoshop, SPSS, Internet Explorer และเกมส์ต่างๆ เป็นต้น

1.3.3 บุคลากร (People ware) หมายถึง บุคลากรในงานด้านคอมพิวเตอร์ซึ่งมีความรู้เกี่ยวกับคอมพิวเตอร์สามารถใช้งาน สั่งงานเพื่อให้คอมพิวเตอร์ทำงานตามที่ต้องการ แบ่งออกได้ 4 ระดับ ดังนี้

1) ผู้จัดการระบบ (System Manager) คือ ผู้วางนโยบายการใช้คอมพิวเตอร์ให้เป็นไปตามเป้าหมายของหน่วยงาน

2) นักวิเคราะห์ระบบ (System Analyst) คือ ผู้ที่ศึกษาระบบงานเดิมหรืองานใหม่และทำการวิเคราะห์ความเหมาะสม ความเป็นไปได้ในการใช้คอมพิวเตอร์กับระบบงาน เพื่อให้โปรแกรมเมอร์เป็นผู้เขียนโปรแกรมให้กับระบบงาน

3) โปรแกรมเมอร์ (Programmer) คือผู้เขียนโปรแกรมสั่งงานเครื่องคอมพิวเตอร์เพื่อให้ทำงานตามความต้องการของผู้ใช้ โดยเขียนตามแผนผังที่นักวิเคราะห์ระบบได้เขียนไว้

4) ผู้ใช้ (User) คือ ผู้ใช้งานคอมพิวเตอร์ทั่วไปซึ่งต้องเรียนรู้วิธีการใช้เครื่อง และวิธีการใช้งานโปรแกรม เพื่อให้โปรแกรมที่มีอยู่สามารถทำงานได้ตามที่ต้องการเนื่องจากเป็นผู้กำหนดโปรแกรม และใช้งานเครื่องคอมพิวเตอร์ มนุษย์จึงเป็นตัวแปรสำคัญในอันที่จะทำให้ผลลัพธ์มีความน่าเชื่อถือ เนื่องจากคำสั่งและข้อมูลที่ใช้ในการประมวลผล ได้รับจากการกำหนดของมนุษย์ทั้งสิ้น

1.3.4 ข้อมูล/สารสนเทศ (Data/Information) ข้อมูล (Data) เป็นองค์ประกอบที่สำคัญอย่างหนึ่ง การทำงานของคอมพิวเตอร์จะเกี่ยวข้องกับข้อมูลตั้งแต่การนำข้อมูลเข้าจนกลายเป็นข้อมูลที่สามารถใช้ประโยชน์ต่อได้หรือที่เรียกว่าสารสนเทศ (Information) ซึ่งข้อมูลเหล่านี้อาจจะเป็นได้ทั้งตัวเลข ตัวอักษร และข้อมูลในรูปแบบอื่นๆ เช่น ภาพ เสียง เป็นต้น ข้อมูลที่จะนำมาใช้กับคอมพิวเตอร์ได้นั้น โดยปกติจะต้องมีการแปลงรูปแบบหรือสถานะให้คอมพิวเตอร์เข้าใจก่อน จึงจะสามารถเอามาใช้งานในการประมวลผลต่างๆ ได้ เราเรียกสถานะนี้ว่า สถานะแบบดิจิตอล ซึ่งมี2 สถานะเท่านั้น คือ เปิด = (1) และ ปิด = (0)

1.4 ประเภทของคอมพิวเตอร์

เครื่องคอมพิวเตอร์แบ่งออกเป็นหลายประเภท ขึ้นอยู่กับเกณฑ์ที่ใช้ในการแบ่งดังนี้

1.4.1 ตามลักษณะการใช้งาน

1) แบบใช้งานทั่วไป (General Purpose Computer) หมายถึงเครื่องประมวลผล ข้อมูลที่มีความยืดหยุ่นในการทำงาน (Flexible) โดยได้รับการออกแบบให้สามารถประยุกต์ใช้ในงานประเภทต่าง ๆ ได้โดยสะดวก โดยระบบจะทำงานตามคำสั่งในโปรแกรมที่เขียนขึ้นมา และเมื่อผู้ใช้ต้องการให้เครื่องคอมพิวเตอร์ทำงานอะไร ก็เพียงแต่ออกคำสั่งเรียกโปรแกรมที่เหมาะสมเข้ามาใช้งาน โดยเราสามารถเก็บโปรแกรมไว้หลายโปรแกรมในเครื่องเดียวกันได้ เช่น ในขณะหนึ่งเราอาจใช้เครื่องนี้ในงานประมวลผลเกี่ยวกับระบบบัญชี และในขณะหนึ่งก็สามารถใช้ใน การออกเช็คเงินเดือนได้เป็นต้น

2) แบบใช้งานเฉพาะด้าน (Special Purpose Computer) หมายถึงเครื่องประมวลผลข้อมูลที่ถูกออกแบบตัวเครื่องและโปรแกรมควบคุม ให้ทำงานอย่างใดอย่างหนึ่งเป็นการเฉพาะโดยทั่วไปมักใช้ในงานควบคุม หรืองานอุตสาหกรรมที่เน้นการประมวลผลแบบรวดเร็วเช่นเครื่องคอมพิวเตอร์ควบคุมสัญญาณไฟจราจร คอมพิวเตอร์ควบคุมลิฟต์ หรือคอมพิวเตอร์ควบคุมระบบอัตโนมัติในรถยนต์เป็นต้น

1.4.2 ตามขนาดและความสามารถ

เป็นการจำแนกประเภทของคอมพิวเตอร์และความสามารถ โดยแบ่งออกได้ดังนี้

1) ซุปเปอร์คอมพิวเตอร์ (Super Computer) หมายถึง เครื่องประมวลผลข้อมูลที่มีความสามารถในการประมวลผลสูงที่สุด โดยทั่วไปสร้างขึ้นเป็นการเฉพาะเพื่องานด้านวิทยาศาสตร์ที่ต้องการการประมวลผลซับซ้อนและต้องการความเร็วสูง เช่น งานวิจัยขีปนาวุธ งานโครงการอวกาศสหรัฐ (NASA) งานสื่อสารดาวเทียม หรืองานพยากรณ์อากาศ เป็นต้น

2) เมนเฟรมคอมพิวเตอร์ (Mainframe computer) เมนเฟรมคอมพิวเตอร์เป็นเครื่องคอมพิวเตอร์ขนาดใหญ่ ทำงานร่วมกับอุปกรณ์หลายๆ อย่างด้วยความเร็วสูงใช้ในงานธุรกิจขนาดใหญ่ มหาวิทยาลัย ธนาคาร และ โรงพยาบาล เมนเฟรมคอมพิวเตอร์สามารถเก็บข้อมูลที่มีปริมาณมากๆ เช่น สามารถเชื่อมโยงใช้งานกับเครื่องเทอร์มินัล (Terminal) หลาย ๆ เครื่อง ในระยะทางไกลกันได้ ระบบเอทีเอ็ม (ATM) การประมวลผลข้อมูลของระบบเมนเฟรมนี้มีผู้ใช้หลาย ๆ คนในเวลาเดียวกัน (Multi-user) สามารถประมวลผลโดยแบ่งเวลาการใช้ซีพียู (CPU) โดยผ่านเครื่องเทอร์มินัลการประมวลผลแบบแบ่งเวลานี้เรียกว่า Time sharing

3) มินิคอมพิวเตอร์ (Mini Computer) โดยทั่วไปธุรกิจและหน่วยงานที่มีขนาดเล็กไม่จำเป็นต้องใช้คอมพิวเตอร์ขนาดเมนเฟรมซึ่งมีราคาแพง ผู้ผลิตคอมพิวเตอร์จึงพัฒนาคอมพิวเตอร์ให้มีขนาดเล็กและมีราคาถูกลง หน่วยงานและบริษัทที่ใช้คอมพิวเตอร์ขนาดนี้ ได้แก่ กรม กอง มหาวิทยาลัย ห้างสรรพสินค้า โรงแรม โรงพยาบาล และโรงงานอุตสาหกรรมต่างๆ

4) ไมโครคอมพิวเตอร์ (Microcomputer) เป็นเครื่องคอมพิวเตอร์ที่มีขนาดเล็กที่สุด ราคาถูกที่สุด ใช้งานง่าย มีขนาดเล็กสามารถตั้งบนโต๊ะ (Desktop) หรือ ใส่ลงในกระเป๋าเอกสาร ไมโครคอมพิวเตอร์สามารถทำงานในลักษณะประมวลผลได้ด้วยตนเอง โดยไม่ต้องเชื่อมโยงกับคอมพิวเตอร์เครื่องอื่น (Standalone system) มีไว้สำหรับใช้งานส่วนตัว จึงเรียกอีกชื่อหนึ่งว่า คอมพิวเตอร์ส่วนบุคคล

5) คอมพิวเตอร์มือถือ (Handheld Computer) เป็นคอมพิวเตอร์ที่มีขนาดเล็กที่สุดเมื่อเทียบกับคอมพิวเตอร์ประเภทอื่นๆ อีกทั้งสามารถพกพาไปยังที่ต่างๆได้ง่ายกว่าเหมาะกับการจัดการข้อมูลประจำวัน การสร้างปฏิทินนัดหมาย การดูหนังฟังเพลง รวมถึงการรับส่งอีเมล์ บางรุ่น อาจมีความสามารถเทียบเคียงได้กับไมโครคอมพิวเตอร์ เช่น Android phone หรือ Apple phone เป็นต้น

1.5 ประโยชน์ของคอมพิวเตอร์

จากการที่คอมพิวเตอร์มีลักษณะเด่นหลายประการ ทำให้ถูกนำมาใช้ประโยชน์ต่อการดำเนินชีวิตประจำวันในสังคมเป็นอย่างมาก ที่พบเห็นได้บ่อยที่สุดก็คือ การใช้ในการพิมพ์เอกสารต่างๆ ซึ่งเรียกว่างานประมวลผล (Word processing) นอกจากนี้ยังมีการประยุกต์ใช้คอมพิวเตอร์ในด้านต่างๆ อีกหลายด้านดังต่อไปนี้

1.5.1 งานธุรกิจ เช่น บริษัท ร้านค้า ห้างสรรพสินค้า ตลอดจนโรงงานต่างๆใช้คอมพิวเตอร์ในการทำบัญชีงานประมวลคำและติดต่อกับหน่วยงานภายนอกผ่านระบบโทรคมนาคม นอกจากนี้งานอุตสาหกรรมส่วนใหญ่ก็ใช้คอมพิวเตอร์มาช่วยในการควบคุมการผลิต และการประกอบชิ้นส่วนของอุปกรณ์ต่าง ๆ เช่น โรงงานประกอบรถยนต์ ซึ่งทำให้การผลิตมีคุณภาพดีขึ้นบริษัทยังสามารถรับ หรืองานธนาคาร ที่ให้บริการถอนเงินผ่านตู้ฝากถอนเงินอัตโนมัติ (ATM) และใช้คอมพิวเตอร์คิดดอกเบี้ยให้กับผู้ฝากเงิน และการโอนเงินระหว่างบัญชีเชื่อมโยงกันเป็นระบบเครือข่าย

1.5.2 งานวิทยาศาสตร์การแพทย์และงานสาธารณสุข สามารถนำคอมพิวเตอร์มาใช้ในนำมาใช้ในส่วนของการคำนวณที่ซับซ้อน เช่น งานศึกษาโมเลกุลสาร เคมี วิถีการโคจรของการส่งจรวดไปสู่อวกาศ หรืองานทะเบียน การเงิน สถิติ และ เป็นอุปกรณ์สำหรับการตรวจรักษาโรคได้ซึ่งจะให้ผลที่แม่นยำกว่าการตรวจด้วยวิธีแบบเดิมทำให้การรักษาได้รวดเร็วขึ้น

1.5.3 งานคมนาคมและสื่อสาร เกี่ยวกับการเดินทาง จะใช้คอมพิวเตอร์ในการจองวันเวลา ที่นั่ง ซึ่งมีการเชื่อมโยงไปยังทุกสถานีทำให้สะดวกต่อผู้เดินทางที่ไม่ต้องเสียเวลารอ อีกทั้งยังใช้ในการควบคุมระบบการจราจร เช่น ไฟสัญญาณจราจร และการจราจรทางอากาศ

1.5.4 งานวิศวกรรมและสถาปัตยกรรม ใช้คอมพิวเตอร์ในการออกแบบหรือจำลองสภาวการณ์ต่างๆ เช่น การรับแรงสั่นสะเทือนของอาคารเมื่อเกิดแผ่นดินไหว รวมทั้งการใช้ควบคุมและติดตามความก้าวหน้าของโครงการต่างๆ เช่น คนงาน เครื่องมือ ผลการทำงาน

1.5.5 งานราชการ เป็นหน่วยงานที่มีการใช้คอมพิวเตอร์มากที่สุด โดยมีการใช้หลายรูปแบบ ทั้งนี้ขึ้นอยู่กับบทบาทและหน้าที่ของหน่วยงานนั้นๆ เช่น ระบบประชุมทางไกลผ่านคอมพิวเตอร์, กรมสรรพากร ใช้จัดการในงานจัดเก็บภาษี บันทึกการเสียภาษีเป็นต้น

1.5.6 การศึกษา ใช้คอมพิวเตอร์ทางด้านการเรียนการสอน ซึ่งมีการนำคอมพิวเตอร์มาช่วยการสอนในลักษณะบทเรียน CAI หรือ งานด้านทะเบียน ซึ่งทำให้สะดวกต่อการค้นหาข้อมูลนักเรียน งานห้องสมุด

1.6 ระบบเครือข่ายคอมพิวเตอร์

ระบบเครือข่ายคอมพิวเตอร์ คือระบบที่มีการนำคอมพิวเตอร์มาต่อเข้าด้วยกันเป็นเครือข่าย เพื่อให้สื่อสารกันและกันได้ โดยทั่วไปจะใช้ระบบ LAN: Local Area Network หรือเครือข่ายท้องถิ่น ซึ่งต้องอาศัยอุปกรณ์เครือข่าย คือ LAN Card, Hub หรือ Switch และสาย UTP จากนั้นก็จะต้องเรื่องของซอฟต์แวร์ในปัจจุบันเราใช้ระบบปฏิบัติการ Windows XP หรือ Windows Vista ซึ่งสนับสนุนการใช้งานเครือข่ายอยู่แล้วก็เหลือเพียงทำการติดตั้งให้ถูกต้อง เช่นการติดตั้งเครื่องพิมพ์ การแชร์ไฟล์ และแชร์เครื่องพิมพ์ เป็นต้น

1.7 อินเตอร์เน็ต

อินเตอร์เน็ต (Internet) คือการเชื่อมต่อคอมพิวเตอร์ให้เป็นระบบเครือข่ายที่ครอบคลุมไปทั่วโลก โดยมีการกำหนดชื่อ และหมายเลข IP ไม่ให้ซ้ำกันในแต่ละเครื่อง และจะมีเครื่องคอมพิวเตอร์บางตัวทำหน้าที่ Server ซึ่งทำหน้าที่ให้บริการด้านต่าง ๆ เช่น Web Server ซึ่งให้บริการ Webpage เป็นต้น และเครื่องอีกส่วนหนึ่งจะถูกติดตั้งให้เป็นเครื่องจะถูกติดตั้งเป็น Client ก็จะเป็นเครื่องที่จะต่อเข้าไปขอใช้บริการจาก เครื่อง Server ต่าง ๆ นั้นเอง การใช้งานอินเตอร์เน็ตนั้นเป็นทราบกันดีว่าเป็นย่อโลกให้เล็กลงเพียงปลายนิ้ว ลดค่าใช้จ่ายในการเดินทาง สามารถค้นหาข้อมูลต่าง ๆ อย่างมากมายไม่มีข้อจำกัดได้อย่างรวดเร็ว

บทที่ 2
การปฏิสัมพันธ์ระหว่างฮาร์ดแวร์และซอฟต์แวร์ และ การประมวลผลอิเล็กทรอนิกส์

2.1 บทนำ

คอมพิวเตอร์มีส่วนประกอบหลัก คือ ฮาร์ดแวร์และซอฟต์แวร์ซึ่งมีการทำงานร่วมกันสอดประสานกันทำให้เกิดการทำงานโดยโต้ตอบกันอย่างไรบ้างเป็นกระบวนการทางไฟฟ้า และ สัญญาณอิเล็กทรอนิกส์ ซึ่งความแตกต่างระหว่างฮาร์ดแวร์ ที่จะทำงานโดยสัญญาณไฟฟ้า และซอฟต์แวร์จะทำงานโดยการเขียนโปรแกรม แต่การที่ทั้งสองสิ่งจะทำงานร่วมกันได้ก็ต้องมี ระบบปฏิบัติการ และ ในฮาร์ดแวร์ทุกชิ้นก็ต้องมีโปรแกรมที่บอกถึงลักษณะเฉพาะ ที่เรียกว่า Driver ทำให้เกิดการสร้างปฏิสัมพันธ์ระหว่างฮาร์ดแวร์และซอฟต์แวร์ ในบทนี้จะช่วยให้คุณเข้าใจหลักการทางเทคนิคทั้งหมดทำให้เข้าใจ ฮาร์ดแวร์และซอฟต์แวร์โต้ตอบกันอย่างไร

2.2 ฮาร์ดแวร์

หมายถึง อุปกรณ์ต่างๆที่สามารถจับต้องได้ ที่ประกอบขึ้นเป็นเครื่องคอมพิวเตอร์ มีลักษณะเป็นโครงร่างสามารถมองเห็นด้วยตา เป็น (รูปธรรม) เช่น จอภาพ คีย์บอร์ด เครื่องพิมพ์ เมาส์ เป็นต้น ซึ่งสามารถแบ่งออกเป็นส่วนต่างๆ ตามลักษณะการทำงานได้ 4 หน่วยคือ หน่วยรับข้อมูล (Input Unit) หน่วยประมวลผลกลาง (Central Processing Unit: CPU) หน่วยแสดงผล (Output Unit) หน่วยเก็บข้อมูลสำรอง (Secondary Storage) โดยอุปกรณ์แต่ละตัวมีหน้าที่การทำงานแตกต่างกันตามจุดประสงค์ สำหรับฮาร์ดแวร์จะติดตั้งในเครื่องคอมพิวเตอร์เพื่อให้ซอฟต์แวร์สั่งการผ่านรหัสโปรแกรม (Code) เพราะฉะนั้นฮาร์ดแวร์และซอฟต์แวร์จึงทำงานร่วมกัน หรือที่เรียกว่า การปฏิสัมพันธ์ระหว่างฮาร์ดแวร์และซอฟต์แวร์ เพื่อดำเนินการทำงานตามที่ผู้ใช้ได้ออกแบบไว้

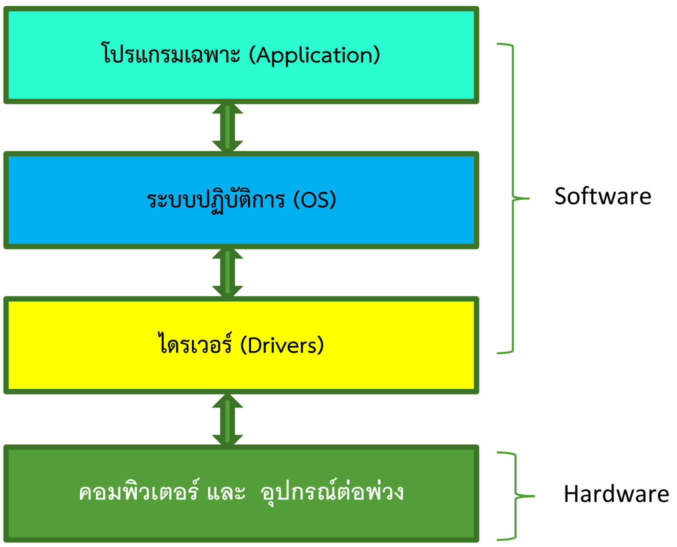
2.3 ซอฟต์แวร์

หมายถึง โปรแกรมหรือชุดคำสั่งหรือ Code ที่ถูกเขียนขึ้นเพื่อสั่งให้ฮาร์ดแวร์คอมพิวเตอร์ทำงานเป็นลำดับขั้นตอน โดยที่ซอฟต์แวร์จึงเป็นเหมือนตัวเชื่อมต่อ ระหว่างผู้ใช้คอมพิวเตอร์กับคอมพิวเตอร์ โดยซอฟต์แวร์สำหรับเครื่องคอมพิวเตอร์เราสามารถแบ่งได้ 3 ประเภท ได้แก่ Driver ของอุปกรณ์ ซอฟต์แวร์ระบบปฏิบัติการ และ ซอฟต์แวร์แอปพลิเคชัน โดยทั่วไปแต่ละโปรแกรมจะมีหน้าที่เฉพาะที่ต้องดำเนินการสำหรับการสร้างโปรแกรม นั่นหมายถึงชุดคำสั่งที่เขียนขึ้นจะมีการปรับเปลี่ยนสถานะการทำงานของฮาร์ดแวร์โดยซอฟท์แวร์ที่ใช้ในการสร้างการปฏิสัมพันธ์ระหว่างฮาร์ดแวร์และซอฟต์แวร์นั้นเป็นภาษาระดับต่ำ หรือภาษาเครื่อง ภาษาแอซเซมบลี

2.4 การปฏิสัมพันธ์ระหว่างฮาร์ดแวร์และซอฟต์แวร์

ฮาร์ดแวร์และซอฟต์แวร์ มีการทำงานที่ประสานงานกัน และเชื่อมโยงในแต่ละยูนิต ตัวอย่างเช่น เมื่อกระทำการเขียนโปรแกรมโดยใช้ภาษาโปรแกรมระดับสูง C, Python, และ C# เมื่อมีการเขียนชุดคำสั่งทั้งหมด และจะถูกแปลงเป็นข้อมูลดิจิตอล ซึ่งเกิดมาจากคอมไพเลอร์มีหน้าที่แปลงรหัสคำสั่งที่เขียนขึ้น เป็นรหัสไบนารี 0, 1 และโดยโปรแกรมจะถูกเขียนตามหลักการของภาษาโปรแกรมนั้นๆ ที่กำหนดไว้ตาม กฎ ไวยากรณ์ โดยข้อมูลตัวเลขไบนารี 0, 1 เหล่านี้จะถูกแปลงเป็น ระดับแรงดันไฟฟ้าเพื่อการควบคุมวงจรไฟฟ้าทั้งหมดที่ผ่านการเรียกว่า ฮาร์ดแวร์

โดยคอมพิวเตอร์ฮาร์ดแวร์ทั้งหมดนั้นมีการทำงานโดยการควบคุมแรงดันไฟฟ้าที่มีความแตกต่างกันส่งสัญญาณควบคุมไปยังวงจรไฟฟ้าโดยแยกเป็นส่วนๆ เพื่อการเปิดใช้งาน หรือ การทำงานเฉพาะอย่าง เช่น การคำนวณ และ การบันทึกข้อมูล สามารถสรุปได้ว่าข้อมูลไบนารีทั้งหมดในคอมพิวเตอร์ เป็นข้อมูลระดับแรงดันไฟฟ้า 0 แรงดันต่ำ และ 1 ระดับแรงดันสูง มีไว้สำหรับควบคุมสั่งการการเปิดหรือปิดการทำงานวงจรโดยการควบคุมของระบบปฏิบัติการ (OS) และ ซอฟต์แวร์ไดรเวอร์ทั้งหมด ช่วยในการปฏิสัมพันธ์ระหว่างฮาร์ดแวร์ และ ซอฟต์แวร์ ทำให้ระบบปฏิบัติการสามารถสื่อสารกับฮาร์ดแวร์ได้จริง และมีการเรียกใช้งาน Application ซอฟต์แวร์ โดยจะแสดงผังการปฏิสัมพันธ์ระหว่างฮาร์ดแวร์และซอฟต์แวร์ดังรูปที่ 2.1



รูปที่ 2.1 การปฏิสัมพันธ์ระหว่างฮาร์ดแวร์และซอฟต์แวร์

จะเห็นได้ว่าในการติดต่อกันมีตัวกลาง Driver ประสานการทำงานระหว่างฮาร์ดแวร์และ ซอฟต์แวร์มีและระบบปฏิบัติการมีหน้าที่จัดการ โปรแกรมต่างๆ ติดต่อประสานงานและทำงาน โดยมีรายละเอียดการทำงานในแต่ละส่วนทำหัวข้อดังนี้

2.5 ไดรเวอร์

ไดรเวอร์ (Driver) หรือ Computer Driver เป็นโปรแกรมคอมพิวเตอร์ที่จะช่วยให้อุปกรณ์ต่างๆ ที่นำมาประกอบเป็นคอมพิวเตอร์ โดยจะมีหน้าที่ขับเคลื่อนให้สามารถทำงานได้ อย่างมีประสิทธิภาพ หรือเชื่อมต่อระหว่างฮาร์ดแวร์ (Hardware) เช่น CPU เมนบอร์ด การ์ดจอ แป้นพิมพ์ เม้าส์ และ ชิ้นส่วนประกอบต่างๆ ของคอมพิวเตอร์ ทำงานร่วมกับระบบปฏิบัติการ (OS) เพื่อให้ระบบปฏิบัติการรู้จักและสามารถใช้งานอุปกรณ์เหล่านั้นได้ ถ้าไม่มีไดรเวอร์ อุปกรณ์คอมพิวเตอร์เหล่านั้นอาจไม่สามารถใช้งานได้ หรืออาจใช้ได้แต่ ไม่เต็มประสิทธิภาพ ดังนั้น และในปัจจุบันมีระบบ Plug and Play คือทุกครั้งที่มีการเพิ่มอุปกรณ์ใหม่ เช่น การ์ดจอใหม่ คอมพิวเตอร์จะไม่สามารถรู้ได้ว่าอุปกรณ์ตัวนั้นชื่ออะไร มีความสามารถทำอะไรได้บ้าง และสามารถสั่งงานอย่างไร โดยจะทำการค้นหาและติดตั้งไดรเวอร์ อัตโนมัติ เพื่อให้คอมพิวเตอร์ทราบถึงรายละเอียดของอุปกรณ์ รวมทั้งวิธีใช้งานอุปกรณ์เหล่านั้นด้วย โดยตัวโปรแกรมนั้นก็คือ ไดรเวอร์ นั่นเอง สำหรับ ระบบปฏิบัติการ Windows จะจัดการกับ Driver ต่าง ๆ เพื่อให้สามารถใช้งานอุปกรณ์เหล่านั้นได้สมบูรณ์

2.6 ไบออส

BIOS ย่อมาจากคำว่า Basic Input Output System เป็นโปรแกรมที่เก็บไว้ในหน่วยความจำถาวร ROM โปรแกรม BIOS เป็นโปรแกรมที่คอมพิวเตอร์เรียกใช้งานเป็นโปรแกรมแรกในการเปิดเครื่อง โดย BIOS จะตรวจสอบ อุปกรณ์ฮาร์ดแวร์ต่างๆ เพื่อความพร้อมโดยการใช้งาน และแจ้งเตือนเมื่อพบปัญหาที่เกิดขึ้นบนอุปกรณ์ต่างๆของฮาร์ดแวร์คอมพิวเตอร์ โดยไบออส ติดตั้งมากับเมนบอร์ดคอมพิวเตอร์

หน้าที่ของ BIOS

BIOS เป็นโปรแกรมขนาดเล็กที่การเรียกการทำงานเป็นโปรแกรมแรกโดย BIOS มีระดับการทำงานระดับต่ำกว่าระบบปฏิบัติการหรือเป็นแกนกลาง มีหน้าที่ตรวจสอบอุปกรณ์ต่างว่ามีการทำงานสมบูรณ์หรือไม่ ตัวอย่างเช่น ในการเชื่อมต่อ ฮาร์ดดิสก์ จอภาพ หรืออุปกรณ์อื่น เพื่อตรวจสอบความผิดพลาดต่างๆ เมื่อพบความผิดพลาดจะรายงานออกมาแจ้งเตือนรูปแบบเสียงในตอนเปิดเครื่อง หรือ ที่หน้าจอภาพ หลังจากตรวจสอบแล้วไบออสจะส่งการทำงานที่ระบบปฏิบัติโดยการอ่านข้อมูลจากฮาร์ดดิสก์ไปที่แรม (RAM) หลังจาก

นั้นไบออสจะทำหน้าที่บริหารจัดการการส่งของข้อมูลระหว่างระบบปฏิบัติการของคอมพิวเตอร์กับอุปกรณ์ที่เชื่อมต่ออยู่

2.7 ระบบปฏิบัติการ

ระบบปฏิบัติการ (Operating System) คือโปรแกรมที่มีการเรียกใช้งานโหลดเข้าสู่อุปกรณ์เครื่องคอมพิวเตอร์หรืออุปกรณ์คอมพิวเตอร์เคลื่อนที่ โดยโปรแกรมทำหน้าที่เริ่มการทำงานระบบ และ มีหน้าที่จัดการประสานงานกับแอปพลิเคชันต่าง ๆ ภายในอุปกรณ์เครื่องคอมพิวเตอร์ให้สามารถทำงานได้โดยแอป-พลิเคชันจะทำงานผ่านระบบปฏิบัติการในการร้องขอบริการ (Service) ต่าง ๆ ที่ถูกเตรียมไว้ให้บริการ ผู้เขียนโปรแกรมสามารถเรียกใช้งานบริการต่างๆ ผ่านระบบปฏิบัติการได้ จากหน้าจอผู้ใช้งาน หรือ หน้าจอที่ติดต่อกับผู้ใช้งาน (User Interface) เช่นในรูปแบบของหน้าจอแบบกราฟิก (Graphical User Interface GUI) ได้

2.7.1 ระบบปฏิบัติการจำเป็นต้องมีเพราะอะไร

ระบบปฏิบัติการมีประโยชน์เพราะว่าอำนวยความสะดวก ทำให้ผู้สร้างซอฟต์แวร์คอมพิวเตอร์ และพัฒนาซอฟต์แวร์ มีการให้บริการ UI (User Interface) และมี ฟังก์ชันการใช้งานเบื้องต้นให้ครอบคลุมได้ทุกเรื่องที่เกี่ยวข้องกับการประมวลผลแอปพลิเคชัน ทำให้ผู้พัฒนาไม่ต้องทำโปรแกรมทุกอย่าง มีหน้าที่จัดการพื้นที่จัดเก็บ, ติดต่อระบบเน็ตเวิร์ก, และอื่นๆ เป็นผลทำให้แอปพลิเคชันมีการเขียนได้ง่ายขึ้น และ Code มีขนาดเล็ก ซึ่งระบบปฏิบัติการนี้สามารถลดภาระการเขียนโค้ด และลดเวลาในการพัฒนาโปรแกรม และ ทำให้ผู้เขียนโปรแกรมสามารถควบคุม และ จัดการฮาร์ดแวร์ระบบ ผ่านการเชื่อมต่อของระบบปฏิบัติการที่มีการใช้งานเข้าใจง่าย

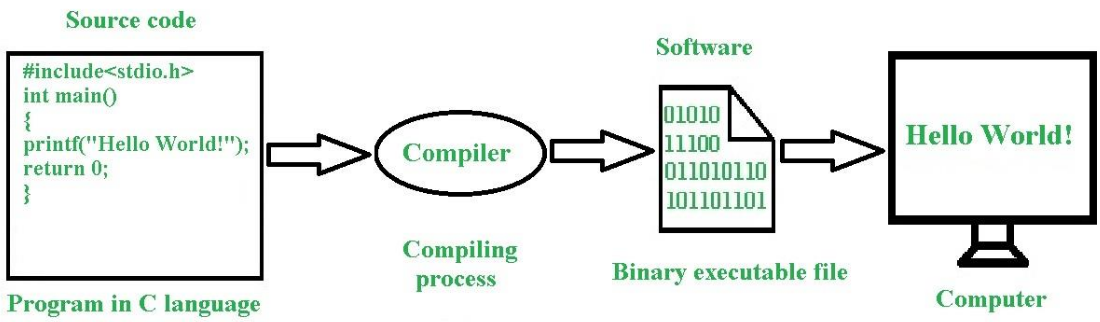
2.7.2 ประเภทของระบบปฏิบัติการ

- ระบบปฏิบัติการคอมพิวเตอร์ (Computer Operating System) สร้างมาเพื่อให้โปรแกรมต่างๆ สามารถทำงานกับฮาร์ดแวร์ที่แตกต่างกันได้อย่างหลากหลาย และทำให้ผู้ใช้งานนั้นสามารถเปิดการทำงานโปรแกรม ได้มากกว่าหนึ่งโปรแกรมอย่างต่อเนื่องและไม่ติดขัด สามารถเปิดการทำงานโปรแกรมได้เช่น การทำบัญชี, การจัดการฐานข้อมูล, ท่องเว็บผ่านเบราว์เซอร์ หรือ การเล่นเกม

- ระบบปฏิบัติการบนมือถือ (Mobile Operating System) มีการออกแบบผู้ใช้อุปกรณ์โทรศัพท์มือถือที่มีการพกพาติดตัวตลอดเวลา และ เป็นอุปกรณ์ตัวกลางสำหรับการติดต่อสื่อสารให้มีความสามารถที่หลากหลายคล้ายกับคอมพิวเตอร์ เช่น สมาร์ทโฟน, จะเห็นได้ว่าอุปกรณ์เคลื่อนที่เหล่านี้มีขนาดเล็กมีทรัพยากร ที่มีอยู่ในเครื่องมือย่างจำกัด เมื่อเทียบกับเครื่องคอมพิวเตอร์ทั่วไปดังนั้นความสามารถจะไม่เท่าคอมพิวเตอร์

- ระบบปฏิบัติการที่ถูกออกแบบเฉพาะเจาะจง (Embedded Operating System) ถูกออกแบบให้กับบางอุปกรณ์มีความจำเป็นต้องการความสามารถที่มากกว่าหนึ่งประการ ใช้การทำงานในรูปแบบที่เฉพาะ เช่น เครื่องกดเงินอัตโนมัติ (ATM), เครื่องคิดเงิน (POS), ฯลฯ โดยเครื่องมือเหล่านี้มีการทำงานเหมือนกับคอมพิวเตอร์ที่สามารถทำงานได้หลายๆ งานในเวลาเดียวกัน ซึ่งมีระบบปฏิบัติการอยู่ภายใน โดยต่างกันตรงจุดประสงค์ในการใช้งาน
- ระบบปฏิบัติการเน็ตเวิร์ก (Network Operating System) ถูกออกแบบมาเพื่อใช้เป็นระบบปฏิบัติการทำหน้าที่สื่อสารระหว่างอุปกรณ์ต่างชนิดกันให้สามารถเชื่อมต่อเครือข่าย โดยทำหน้าที่เชื่อมต่อให้การสื่อสารที่ระหว่างจุดสองจุดให้มีการทำงานสมบูรณ์

รูปแบบการปฏิสัมพันธ์ระหว่างฮาร์ดแวร์และซอฟต์แวร์



รูปที่ 2.2 ตัวอย่างการปฏิสัมพันธ์ระหว่างซอฟต์แวร์ สู่ ฮาร์ดแวร์

แหล่งที่มารูปภาพ https://www.geeksforgeeks.org/what-is-a-computer-software/

จากภาพที่ 2.2 อธิบายได้ว่าเริ่มจากนักพัฒนาโปรแกรมออกแบบคำสั่งหรือผู้ใช้ทำการสั่งงานเครื่องคอมพิวเตอร์โดยภาพตัวอย่างใช้โปรแกรมภาษาซี โดยเครื่องไม่สามารถเข้าใจ Code ได้เพราะเครื่องเป็นอุปกรณ์อิเล็กทรอนิกส์ มนุษย์จึงออกแบบตัวกลางนั่นคือ Compiler ทำหน้าที่แปลความหมายร่วมกับ โปรแกรมระบบปฏิบัติการ และผลการแปลที่ได้จะได้ภาษาเครื่องซึ่งภาษาเครื่องนี้จะเป็นรหัสไบนารี เลขฐานสอง ที่มีระดับแรงดัน 2 ระดับ ลอจิกแรงดัน 0 และ 1 และเมื่อมีการแปลงแล้วจากนั้นการทำงานถูกส่ง ไปยังอุปกรณ์ที่ต้องการสั่งงาน ดังตัวอย่างคือการแสดงผลออกทางจอภาพ ในขั้นตอนนี้เป็นการทำงานร่วมกันระหว่าง ระบบปฏิบัติการ ไดรเวอร์ และไบออส เพื่อให้ตีความหมายคำสั่งไปยังอุปกรณ์ที่ต้องการให้มีการทำงานเป็นรหัสไบนารี และมีการกำหนดตำแหน่งที่อยู่ฮาร์ดแวร์ และ ข้อมูลที่ต้องการแสดงผลออกทางหน้าจอ ก็คือ รหัสคำสั่งจะถูกแปลให้อยู่ในรูปแบบของสัญญาณไฟฟ้าเพื่อสั่งงานวงจรอิเล็กทรอนิกส์ทำงานตามที่โปรแกรมเมอร์ออกแบบ รหัสคำสั่งที่ต้องการนั่นเอง

2.8 การประมวลผลอิเล็กทรอนิกส์

การนับของมนุษย์โดยปกตินั้น มีตัวเลขอยู่สิบตัวคือ 0, 1, 2, 3, 4, 5, 6, 7, 8, และ 9 ซึ่งจะประกอบรวมกัน เรียกกันว่า “เลขฐาน 10”

เช่น ตัวเลข 7,392

$$7,392 = (7 \times 10^3) + (3 \times 10^2) + (9 \times 10^1) + (2 \times 10^0)$$

$$= 7000 + 300 + 90 + 2$$

เมื่อ	=	
10^0	1	หลักหน่วย
	=	
	1	
10^1	0	หลักสิบ
	=	
	1	
	0	
10^2	0	หลักร้อย
	=	
	1	
	0	
	0	
10^3	0	หลักพัน

1.6.1 ส่วนประกอบของเลขฐานต่าง ๆ

ระบบตัวเลขที่นิยมใช้กับเครื่องคอมพิวเตอร์มีอยู่ 2 ระบบคือ เลขฐานสอง และ เลขฐานสิบหก ซึ่งเลขแต่ละฐานจะมีตัวเลขไม่เท่ากันดังรายละเอียดต่อไปนี้

1.6.1.1 เลขฐานสอง ประกอบด้วยเลข 2 ตัวคือ 0 และ 1 เท่านั้น

1.6.1.2 เลขฐานแปด ประกอบด้วยเลข 8 ตัวคือ 0 , 1 , 2 , 3 , 4 , 5 , 6 และ 7 เท่านั้น

1.6.1.3 เลขฐานสิบหก ประกอบด้วยเลข 16 ตัวคือ 0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , A , B , C , D , E และ F เท่านั้น เนื่องจากจำนวนนับ มี 0-9 ดังนั้นเพื่อความเข้าใจ A ในที่นี้ก็คือ 10 ตัว B มีค่าเท่ากับ 11 ตัว C มีค่าเท่ากับ 12 ตัว D มีค่าเท่ากับ 13 ตัว E มีค่าเท่ากับ 14 และตัว F มีค่าเท่ากับ 15

1.6.2 การแปลงเลขฐานสอง เลขฐานแปด เลขฐานสิบหก เป็นเลขฐานสิบ

การแปลงเลขฐานอื่น ๆ เป็นเลขฐานสิบมีหลักเกณฑ์ดังต่อไปนี้

$$N = d_nR^n + \ldots + d_3R^3 + d_2R^2 + d_1R^1 + d_0R^0 + d_{-1}R^{-1} + d_{-2}R^{-2} + d_{-3}R^{-3} + \ldots.$$

<u>ตัวอย่างที่</u> 1.1 การแปลง $(1101)_2$ ให้เป็นเลขฐานสิบ

<u>วิธีทำ</u>

$N = (1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0)$

$= 8 + 4 + 0 + 1$

$= 13$

<u>ตอบ</u> $(1101)_2 = 13$

<u>ตัวอย่างที่</u> 1.2 การแปลง $(1011)_2$ ให้เป็นเลขฐานสิบ

<u>วิธีทำ</u> $N = (1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0)$

$= 8 + 0 + 2 + 1$

$= 11$

<u>ตอบ</u> $(1011)_2 = 11$

<u>ตัวอย่างที่</u> 1.3 การแปลง $(7C9)_{16}$ ให้เป็นเลขฐานสิบ

วิธีทำ $N = (7 \times 16^2 + C \times 16^1 + 9 \times 16^0)$

$= (7 \times 256) + (12 \times 16) + (9 \times 1)$

$= (1792 + 192 + 9$

<u>ตอบ</u> $(7C9)_{16} = 1993$

1.6.3 การแปลงเลขฐานสิบให้เป็นเลขฐานสอง เลขฐานสิบหก

การแปลงเลขฐานสิบไปเป็นเลขฐานต่างๆ มีหลักเกณฑ์ดังต่อไปนี้

1) ให้นำเลขฐานสิบตั้ง แล้วหารด้วยตัวเลขฐาน เศษที่ได้คือผลลัพธ์ของเลขฐานที่จะแปลง

2) นำผลลัพธ์ที่ได้ ตั้งแล้วหารด้วยตัวเลขฐาน ต่อไปเรื่อย ๆ จนกระทั่งผลลัพธ์มีค่าเป็นศูนย์ การหารแต่ละครั้งเศษที่ได้ จะเป็นผลลัพธ์ของเลขฐานเลขฐานที่จะแปลง

3) เศษที่ได้จากการหารนั้น เศษตัวแรกจะเป็นบิตแรกของเลขฐานเลขฐานที่จะแปลง และเศษตัวสุดท้ายจะเป็นบิตสุดท้ายของเลขฐานเลขฐานที่จะแปลง ดังตัวอย่างต่อไปนี้

<u>ตัวอย่างที่</u> 1.4 การแปลง $(27)_{10}$ ให้เป็นเลขฐานสอง

<u>วิธีทำ</u> $(27)_{10} = (\ldots\ldots\ldots\ldots\ldots)_2$

หารครั้งที่ 1 $27 \div 2 = 13$ เศษ 1

หารครั้งที่ 2 $13 \div 2 = 6$ เศษ 1

หารครั้งที่ 3 $6 \div 2 = 3$ เศษ 0

หารครั้งที่ 4 $3 \div 2 = 1$ เศษ 1

หารครั้งที่ 5 $1 \div 2 = 0$ เศษ 1

<u>ตอบ</u> $27 = (11011)_2$

<u>ตัวอย่างที่</u> 1.5 การแปลง $(520)_{10}$ ให้เป็นเลขฐาน 16

<u>วิธีทำ</u> $(520)_{10} = (\ldots\ldots\ldots\ldots\ldots)_8$

หารครั้งที่ 1 $520 \div 16 = 32$ เศษ 8

หารครั้งที่ 2 $32 \div 16 = 2$ เศษ 0

<u>ตอบ</u> $520 = (208)_{16}$

1.6.4 การแปลงเลขฐานสิบที่เป็นทศนิยมให้เป็นเลขฐานสอง

การแปลงเลขฐานสิบที่เป็นทศนิยมไปเป็นเลขฐานต่างๆ มีหลักเกณฑ์ดังต่อไปนี้

1) นำตัวเลขเฉพาะจุดทศนิยมเท่านั้นมาตั้ง แล้วคูณด้วยตัวเลขฐาน ผลลัพธ์ที่อยู่หน้าจุดทศนิยมคือผลลัพธ์ของเลขฐานที่จะแปลง

2) นำผลลัพธ์ของการคูณที่อยู่หลังจุดทศนิยมตั้ง แล้วคูณด้วยตัวเลขฐาน ทำเช่นนี้ไปเรื่อย ๆ จนกระทั่งตัวเลขหลังจุดทศนิยมเป็นศูนย์ หรือมีทศนิยมเท่ากับจำนวนที่โจทย์กำหนด

3) เลขหน้าจุดทศนิยมของการคูณครั้งแรกจะเป็นผลลัพธ์ของเลขฐานที่จะแปลงบิตแรก และเลขหน้าจุดทศนิยมของการคูณครั้งสุดท้ายจะเป็นผลลัพธ์ของเลขฐานที่จะแปลงบิตสุดท้าย

<u>ตัวอย่างที่</u> 1.6 การแปลง $(0.125)_{10}$ ให้เป็นเลขฐานสอง

<u>วิธีทำ</u> $(0.125)_{10} = (.................)_2$

คูณครั้งที่ 1 $0.125 \times 2 = 0.250$

คูณครั้งที่ 2 $0.250 \times 2 = 0.500$

คูณครั้งที่ 3 $0.500 \times 2 = 1.000$

<u>ตอบ</u> $0.125 = (0.001)_2$

บทที่ 3
การพัฒนาโปรแกรม ผังงาน

3.1 การวิเคราะห์ปัญหาสำหรับการออกแบบโปรแกรม

ขั้นตอนในการวิเคราะห์ปัญหาต้องเริ่มต้นจากนำปัญหาของงานมาทำการสร้างเป็นขั้นตอนวิธี (Algorithm) ตั้งแต่ขั้นตอนแรกจนถึงขั้นตอนสุดท้าย ในรูปแบบผังงาน (Flow chart) หรือรหัสเทียม (Pseudo-code) แล้วจึงนำขั้นตอนวิธีที่ได้นั้นมาเขียนเป็นภาษาคอมพิวเตอร์ต่อไป

ขั้นตอนของการแก้ปัญหา (Problem Solving) สำหรับพัฒนาการเขียนโปรแกรมคอมพิวเตอร์มี 4 ขั้นตอนดังนี้

3.1.1 วิเคราะห์ปัญหา (Problem Analysis) ระบุปัญหา และทำการวิเคราะห์ปัญหา โดยทำการวิเคราะห์รายละเอียดต่าง ๆ ดังนี้

1) วัตถุประสงค์หลัก (Main purpose) เป็นการกำหนดขอบเขตแบบกว้าง ๆ
2) รูปแบบผลลัพธ์ (Output requirement) เป็นการกำหนดรูปแบบ หรือลักษณะของการแสดงผลลัพธ์
3) ข้อมูลนำเข้า (Input requirement) ปัญหาบอกข้อมูลอะไรบ้าง กำหนดรูปแบบของข้อมูลที่รับเข้า
4) ตัวแปรที่ใช้ (Variable) ได้แก่ ชื่อที่ใช้เป็นตัวแทนข้อมูลทีนำเข้า เรียกว่า ชื่อตัวแปร (Variable Name) ซึ่งการกำหนดชื่อตัวแปรต้องตรงตามเงื่อนไขการตั้งชื่อตัวแปรของแต่ละภาษาที่ใช้เขียนโปรแกรมและสอดคล้องกับชนิดข้อมูล เพื่อง่ายต่อการแก้ไข หรือพัฒนาปรับปรุงโปรแกรม
5) วิธีการประมวลผล (Process) ได้แก่ กำหนดค่า กำหนดสูตร หรือวิธีการกำหนดเงื่อนไขของการคำนวณ เพื่อป้องกันไม่ให้โปรแกรมทำงานผิดพลาด เช่น กรณีการหารด้วยศูนย์ การวนซ้ำไม่รู้จบ
6) ภาษาที่ใช้ (Programming Language) ได้แก่ ภาษาที่ใช้เขียนโปรแกรม

3.1.2 การออกแบบโปรแกรม (Program Design) การออกแบบโปรแกรมเพื่อแก้ปัญหา หรือออกแบบขั้นตอนวิธี (Algorithm Design)

นำผลการวิเคราะห์จากขั้นตอนที่ 1 การวิเคราะห์ปัญหา มาแสดงรายละเอียดเป็นขั้นตอนว่าจะต้องเขียนโปรแกรมเพื่อแก้ปัญหาอย่างไร โดยทำการเขียนเป็นข้อ ๆ ตั้งแต่ขั้นตอนแรกจนถึงขั้นตอนสุดท้าย กระบวนการนี้เรียกว่า **ขั้นตอนวิธี (Algorithm)** ซึ่งแบ่งออกเป็น 2 รูปแบบ คือ

1) ผังงาน (Flow charts) คือ การเขียนขั้นตอนวิธีด้วยรูปภาพ หรือสัญลักษณ์

2) รหัสเทียม (Pseudo code) คือ การเขียนขั้นตอนวิธีด้วยภาษาที่ใกล้เคียงภาษาคอมพิวเตอร์

3.1.3 การเขียนโปรแกรม (Program Coding) เลือกภาษาในการเขียนโปรแกรม และเขียนโปรแกรมจากขั้นตอนวิธีที่ออกแบบไว้ ซึ่งเป็นการนำส่วนที่ได้ออกแบบไว้จากขั้นตอนที่ 2 (ผังงาน หรือรหัสเทียม) มาเขียนเป็นภาษาคอมพิวเตอร์

3.1.4 ทดสอบโปรแกรม และแก้ข้อผิดพลาด (Program Testing and Debugging) ตรวจสอบความถูกต้องของโปรแกรม เพื่อหาข้อผิดพลาดที่อาจเกิดขึ้นได้โดยการทดสอบนั้นจะต้องใช้ข้อมูลหลาย ๆ ชุดข้อมูลในการทดสอบ เพื่อให้ครอบคลุมการทำงานทั้งหมด และเป็นการยืนยันว่าที่ทำการเขียนขึ้นนั้นมีความถูกต้อง

แบ่งการตรวจสอบออกเป็น 3 ส่วนคือ

1) **ความผิดพลาดทางไวยากรณ์ (Syntax Error)** เป็นข้อผิดพลาดที่เกิดเนื่องจากการเขียนคำสั่งของภาษาไม่ถูกต้อง ไม่ถูกตามหลักไวยากรณ์ หรือใช้คำสั่งผิดรูปแบบ ข้อผิดพลาดลักษณะนี้โปรแกรมจะหยุดการทำงาน และปรากฏข้อความแสดงข้อผิดพลาด (Error Message) ทันทีหลังจากโปรแกรมเริ่มทำงาน (Execute Program)
2) **ความผิดพลาดในการดำเนินงาน (Runtime Error)** เป็นข้อผิดพลาดขณะการแปลโปรแกรมที่เกิดขึ้นจากทำคำสั่งและไม่สามารถประมวลผลได้ โปรแกรมจะหยุดและแจ้งความผิดพลาดให้ทราบ
3) **ความผิดพลาดทางตรรกะ (Logical Error)** ได้แก่ ข้อผิดพลาดที่เกิดจากการกำหนดเงื่อนไขของคำสั่งไม่ถูกต้อง เช่น การกำหนดค่าตัวแปรผิด หรือใส่ตัวดำเนินการทางคณิตศาสตร์ผิด เป็นต้น ข้อผิดพลาดนี้โปรแกรมจะไม่หยุดการทำงาน แต่จะให้ผลลัพธ์ที่ไม่ถูกต้องตามเงื่อนไขที่ต้องการเท่านั้น การตรวจสอบและแก้ไขข้อผิดพลาดนี้กระทำได้โดยการทดสอบการทำงานของโปรแกรมว่าถูกต้องเป็นไปตามที่ต้องการหรือไม่

3.2 การเขียนขั้นตอนวิธีแบบผังงาน (Flowchart)

ผังงาน คือ การเขียนอธิบายขั้นตอนรายละเอียดของการทำงานแต่ละส่วนงาน ตั้งแต่ขั้นตอนการรับข้อมูล ขั้นตอนการประมวลผล จนถึงขั้นตอนการแสดงผลลัพธ์ ในลักษณะของรูปภาพ หรือสัญลักษณ์

3.2.1 ประโยชน์ของผังงาน

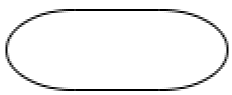
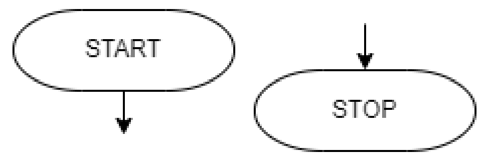
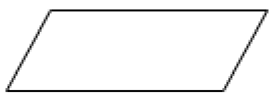
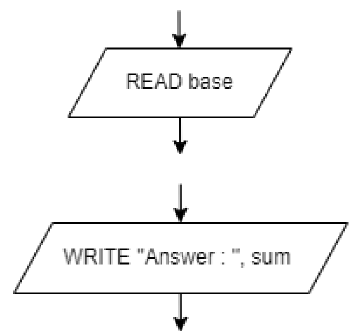
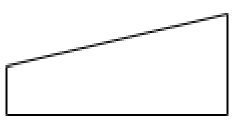
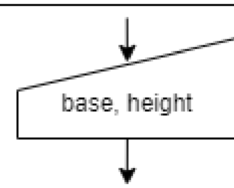
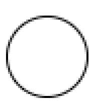
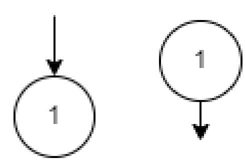
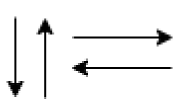
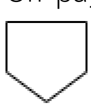
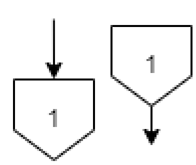
1) แสดงขั้นตอนวิธีการทำงานของแต่ละส่วนตามลำดับก่อนหลังได้ชัดเจนกว่าข้อความ

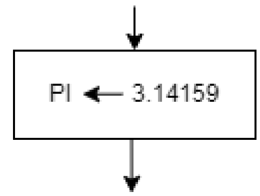
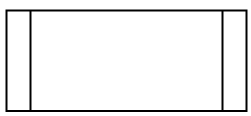
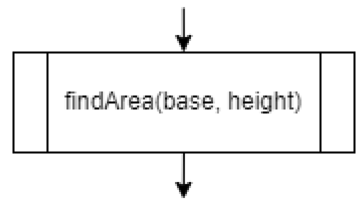
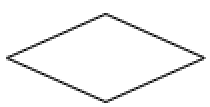
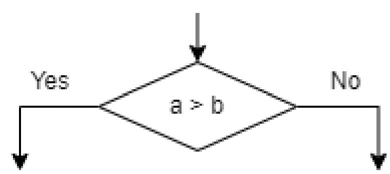
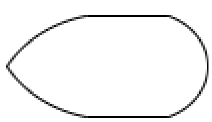
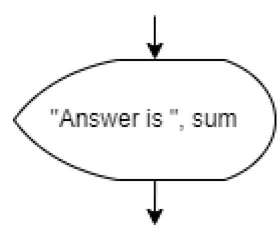
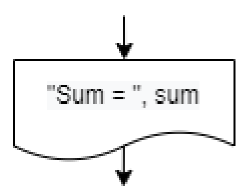
2) ตรวจสอบข้อผิดพลาดของโปรแกรมได้ง่ายกว่าการตรวจสอบจากตัวโปรแกรมโดยตรง

3) สามารถทำความเข้าใจโปรแกรมที่ผู้อื่นเขียนได้ง่ายขึ้น

3.2.2 เครื่องหมายสัญลักษณ์ที่ใช้ในการเขียนผังงาน มีรายละเอียดแสดงดังตารางที่ 3.1

ตารางที่ 3.1 สัญลักษณ์ที่สำหรับการเขียนผังงาน

สัญลักษณ์	ความหมาย	ตัวอย่าง
Terminal 	การเริ่มต้น (Start) และการสิ้นสุด (Stop)	
Input/Output 	การรับ-ส่งข้อมูล (Input and Output) แบบไม่ระบุชนิดอุปกรณ์	
Manual Input 	การรับข้อมูลเข้าโดยใช้แป้นพิมพ์ (Keyboard)	
On-page Connector 	จุดต่อภายในหน้าเดียวกัน	
Flow line/Direction 	ลูกศรแสดงทิศทางการทำงานของผังงาน	
Off-page Connector 	จุดต่อระหว่างหน้า หรือคนละหน้า	

สัญลักษณ์	ความหมาย	ตัวอย่าง
Process 	การกำหนดค่า การคำนวณ และการประมวลผลทั่วไป	
External Subroutine 	การประมวลผลข้อมูลด้วยโปรแกรมย่อยที่อยู่ภายนอกโปรแกรมหลัก	
Preparation 	การเตรียมข้อมูล ใช้กำหนดชื่อและรูปแบบของข้อมูล	
Decision 	การตัดสินใจ การเปรียบเทียบ จะมีผลใน 2 ทิศทาง คือ กรณีผลตรวจสอบเงื่อนไข เป็นเท็จ และเป็นจริง	
Visual Display 	การแสดงผลข้อมูลออกทางจอภาพ	
Document 	การแสดงผลเอกสาร หรือการแสดงผลข้อมูลออกทางเครื่องพิมพ์	

3.2.3 ข้อกำหนดของการเขียนผังงาน

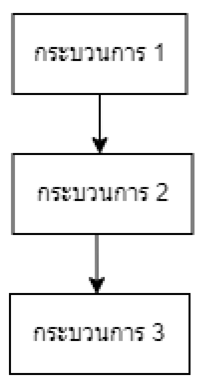
1) ทุกผังงานต้องมีจุดเริ่มต้นและจุดสิ้นสุดอย่างละหนึ่งแห่งเท่านั้น

2) เขียนผังงานจากบนลงล่าง จากซ้ายไปขวา โดยใช้ลูกศรเป็นตัวแสดงเส้นทางการดำเนินงาน
3) ทุกสัญลักษณ์ของผังงานต้องมีลูกศรชี้ทิศทางเข้า และลูกศรชี้ทิศทางออกอย่างละหนึ่งลูกศร ยกเว้นสัญลักษณ์จุดเริ่มต้น จุดสิ้นสุด การตัดสินใจ และจุดต่อ
4) ลูกศรที่ใช้บอกทิศทางไม่ควรเขียนตัดกัน หรือทับกัน หากจำเป็นควรใช้จุดต่อแทน
5) ไม่ควรเขียนเส้นเชื่อมโยงที่อยู่ห่างกันมาก ควรใช้สัญลักษณ์จุดต่อแทน
6) ขนาดของสัญลักษณ์ผังงานตามมาตรฐาน คือ ความยาวเป็นสองเท่าของความกว้าง
7) ข้อความ หรือชื่องานที่ปรากฏอยู่ในสัญลักษณ์ ควรใช้ข้อความที่กระชับและได้ใจความ ไม่จำเป็นต้องแสดงข้อความที่เป็นความหมายของสัญลักษณ์นั้น ๆ
8) นิยมใช้เครื่อง หมายลูกศร (⟵) แทนการใช้เครื่องหมายเท่ากับ (=)
9) ทุกสัญลักษณ์ต้องเชื่อมต่อกับสัญลักษณ์อื่นเสมอ เพราะทุกขั้นตอนมีความเกี่ยวเนื่องกัน

3.2.4 ลักษณะโครงสร้าง Flowchart

รูปแบบพื้นฐานของออกแบบโครงสร้างของโปรแกรมมี 3 แบบ ด้วยกันคือ

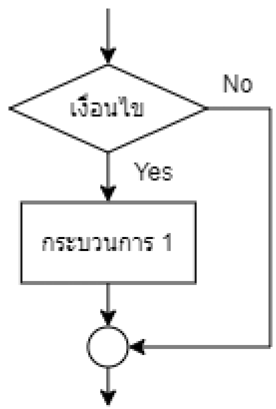
1) โครงสร้างแบบลำดับ (Sequence Structure) เป็นโครงสร้างที่มีการทำงานตามลำดับ เริ่มจากคำสั่งที่ 1 แล้วจึงทำคำสั่งที่ 2 และทำคำสั่งถัดไป (ถ้ามี) จนหมดแสดงดังรูปที่ 3.1



รูปที่ 3.1 ผังงานโครงสร้างแบบลำดับ

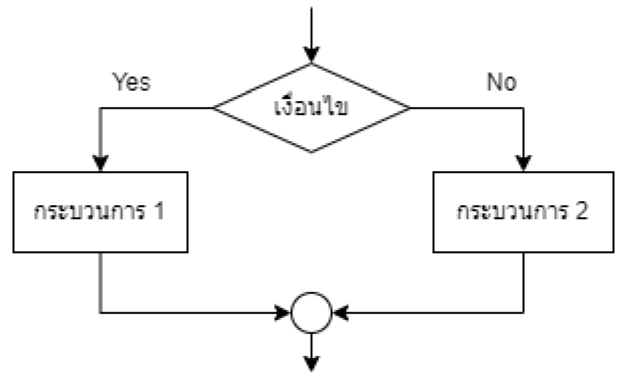
2) โครงสร้างแบบทางเลือก (Selection Structure) เป็นโครงสร้างที่มีการตรวจสอบเงื่อนไขและมีทางเลือกให้เลือกทำอย่างใดอย่างหนึ่ง เช่น if, if…else, switch…case โดยมีรูปผังงานแสดงดังรูปที่ 3.2

การเลือกแบบหนึ่งเส้นทาง



จะทำงานเมื่อเงื่อนไขเป็นจริงเท่านั้น

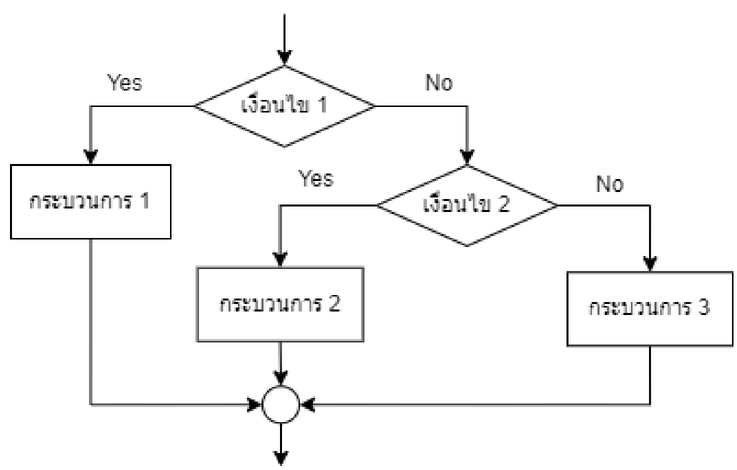
การเลือกแบบสองเส้นทาง



ต้องมีการทดสอบเงื่อนไขก่อน แล้วเลือกกระทำ ถ้าเงื่อนไขเป็นจริงจะทำกระบวนการ 1 แต่ถ้าเงื่อนไขเป็นเท็จจะทำกระบวนการ 2

รูปที่ 3.2 ผังงานโครงสร้างแบบทางเลือก

การเลือกทำแบบเชิงซ้อน (Nested if)

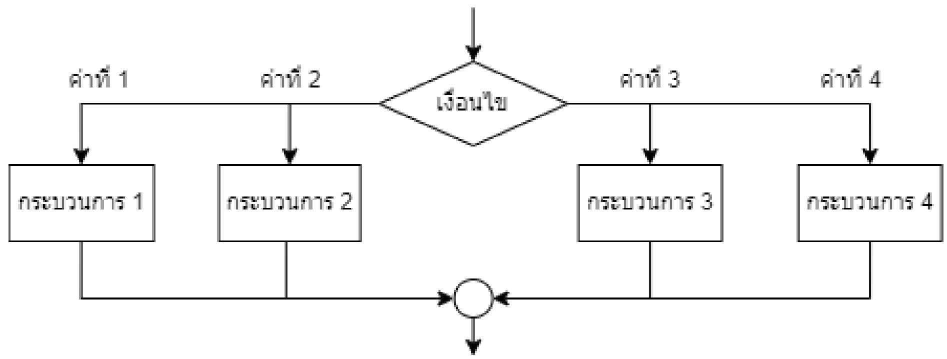


รูปที่ 3.3 ผังงานเลือกทำแบบเชิงซ้อน

จากรูปที่ 3.3 จะเห็นได้ว่าต้องมีการทดสอบเงื่อนไขก่อน แล้วเลือกกระทำ ถ้าเงื่อนไขเป็นจริงจะทำกระบวนการ 1 ถ้าเงื่อนไขเป็นเท็จจะทดสอบเงื่อนไขถัดไป ถ้าเงื่อนไขเป็นจริงทำกระบวนการ 2 ถ้าเงื่อนไข

เป็นเท็จจะตรวจสอบเงื่อนไขอื่นอีกเรื่อย ๆ จนกว่าจะพบเงื่อนไขที่เป็นจริง และทำกระบวนการนั้น ๆ หรือถ้าไม่พบก็จะทำกระบวนการ x ในเงื่อนไขสุดท้าย (ELSE) ก่อนจบการทำงาน

การเลือกทำแบบหลายเส้นทาง (CASE Structure)

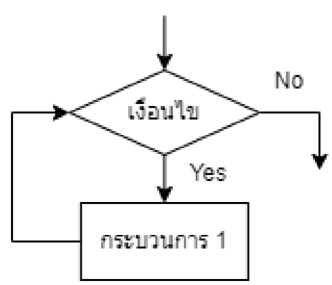


รูปที่ 3.4 ผังงานแบบหลายเส้นทาง

จากรูปที่ 3.4 กรณีที่ต้องมีการเลือก ถ้าเงื่อนไขไหนเป็นจริงจะทำตามกระบวนการหนึ่งในหลาย ๆ ทางเลือก

3) **โครงสร้างแบบทำซ้ำ (Repetition Structure)** เป็นโครงสร้างที่ต้องทำงานใดงานหนึ่งมากกว่า 1 ครั้ง เช่น while-do, repeat-until, for

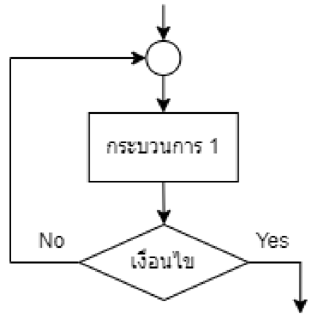
ทำซ้ำในขณะที่ (While-Do)



รูปที่ 3.5 ผังงานแบบทำซ้ำในขณะที่

จากรูปที่ 3.5 มีหลักการดังต่อไปนี้ ขั้นตอนแรกของการทำงาน คือ การตัดสินใจ หากเงื่อนไขเป็นจริง จะทำคำสั่งที่ 1 ซึ่งเรียกว่า “ขั้นตอนการทำซ้ำ” เมื่อทำเสร็จจะกลับไปตรวจสอบเงื่อนไขอีกครั้ง หากเงื่อนไขเป็นเท็จจะออกจากขั้นตอนการทำซ้ำ เพื่อทำงานอื่นต่อไป

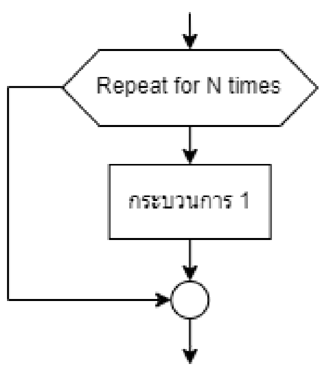
ทำซ้ำจนกระทั่ง (Repeat-Until)



รูปที่ 3.6 ผังงานแบบทำซ้ำจนกระทั่ง

จากรูปที่ 3.6 มีหลักการดังต่อไปนี้ ขั้นตอนแรกของการทำงาน คือ ทำคำสั่งที่ 1 ซึ่งเรียกว่า "ขั้นตอนการทำซ้ำ" ถัดมาจะเป็นการตรวจสอบเงื่อนไข หากเงื่อนไขเป็นเท็จ จะกลับไปทำขั้นตอนการทำซ้ำอีก แต่ถ้าหากเงื่อนไขเป็นจริง จะออกจากขั้นตอนการทำซ้ำ เพื่อทำงานอื่นต่อไป

ทำซ้ำแบบ For



รูปที่ 3.7 ผังงานแบบกำหนดรอบ

จากรูปที่ 3.7 มีหลักการดังต่อไปนี้ โดยผังงานนี้เป็นโครงสร้างที่กำหนดจำนวนให้ทำการวนซ้ำที่แน่นอน แตกต่างจาก 2 รูปแบบแรกที่กล่าวมาแล้วที่ไม่จำเป็นต้องทราบจำนวนครั้งก่อน แต่ใช้การตรวจสอบเงื่อนไขหยุดทำงานเมื่อทำครบ

ตารางที่ 3.2 ความแตกต่างระหว่างการทำซ้ำในขณะที่และการทำซ้ำจนกระทั่ง

	การทำซ้ำในขณะที่ (While-Do)	**การทำซ้ำจนกระทั่ง** (Repeat-Until)
การตรวจสอบเงื่อนไข	ตรวจสอบก่อนทำขั้นตอนการทำซ้ำ	ตรวจสอบหลังทำขั้นตอนการทำซ้ำ
การตัดสินใจเพื่อทำซ้ำ	ทำซ้ำเมื่อผลการตรวจสอบเงื่อนไขเป็น**จริง**	ทำซ้ำเมื่อผลการตรวจสอบเงื่อนไขเป็น**เท็จ**
จำนวนการทำซ้ำ	อาจไม่มีการทำขั้นตอนการทำซ้ำ หากผลการตรวจสอบเงื่อนไขเป็นเท็จตั้งแต่แรก	ทำขั้นตอนการทำซ้ำอย่างน้อย 1 ครั้ง

3.3 ตัวอย่างการประยุกต์ใช้งานผังงาน

ตัวอย่างที่ 1 หาพื้นที่วงกลมและแสดงผลลัพธ์

ข้อมูลนำเข้า : รัศมีของวงกลม

ตัวแปรที่ใช้ : radius แทน รัศมีของวงกลม และ area แทน พื้นที่วงกลม

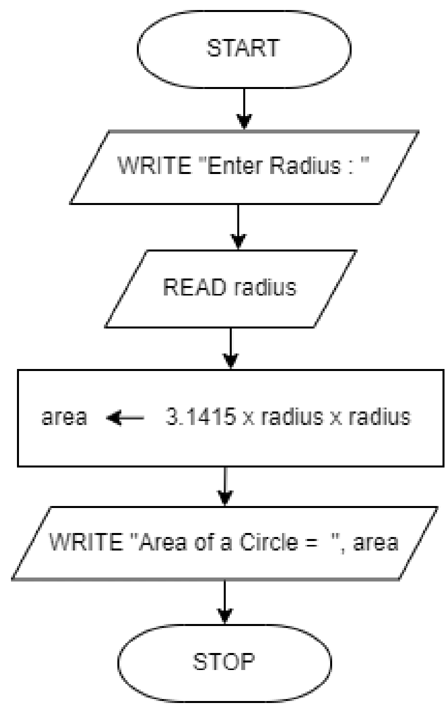
การประมวลผล : area คำนวณได้จากสูตร 3.1415 x รัศมี x รัศมี

ข้อมูลส่งออก (ผลลัพธ์) : พื้นที่วงกลม

ขั้นตอนวิธี

1. เริ่มต้น
2. แสดงข้อความ “Enter Radius : ”
3. รับข้อมูลเข้า แล้วมาเก็บไว้ในตัวแปร radius
4. คำนวณค่าพื้นที่วงกลม จากสูตร 3.1415 x radius x radius แล้วนำค่าที่ได้เก็บไว้ในตัวแปร area
5. แสดงข้อความ “Area of a Circle = “
6. แสดงค่าในตัวแปร area
7. จบการทำงาน

ผังงาน



ตัวอย่างที่ 2 หาพื้นที่สามเหลี่ยมและแสดงผลลัพธ์

ข้อมูลนำเข้า : ความยาวฐาน และความสูง

ตัวแปรที่ใช้ : Base แทน ความยาวฐาน

Height แทน ความสูง

Area แทน พื้นที่สามเหลี่ยม

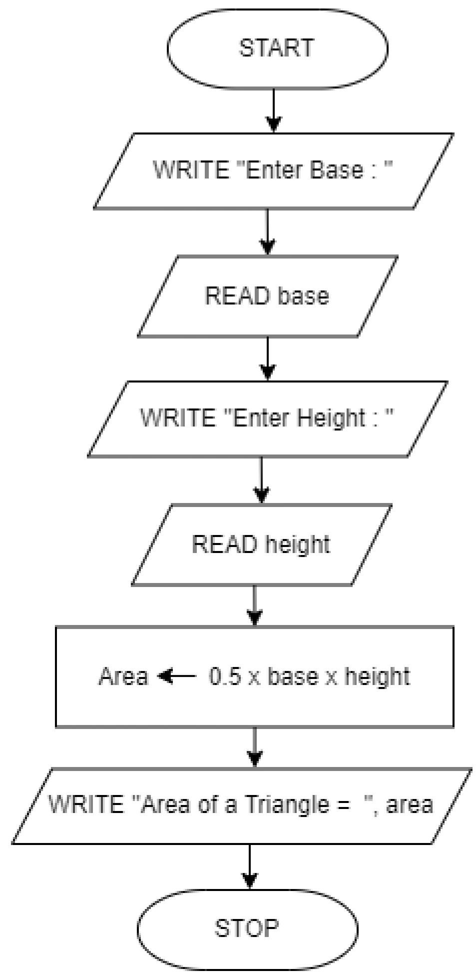
การประมวลผล : Area คำนวณได้จากสูตร 0.5 x ความยาวฐาน x ความสูง

ข้อมูลส่งออก (ผลลัพธ์) : พื้นที่สามเหลี่ยม

ขั้นตอนวิธี

1. เริ่มต้น
2. แสดงข้อความ “Enter Base : ”
3. รับข้อมูลเข้า แล้วมาเก็บไว้ในตัวแปร Base
4. แสดงข้อความ “Enter Height : ”
5. รับข้อมูลเข้า แล้วนำมาเก็บไว้ในตัวแปร Height
6. คำนวณค่าพื้นที่สามเหลี่ยม จากสูตร 0.5 x Base x Height แล้วนำค่าที่ได้เก็บไว้ในตัวแปร Area
7. แสดงข้อความ “Area of a Triangle = ”
8. แสดงค่าในตัวแปร Area
9. จบการทำงาน

ผังงาน



ตัวอย่างที่ 3 รับเลขจำนวนเต็มมาหนึ่งจำนวน และแสดงข้อความนั้นเป็นจำนวนคู่ หรือจำนวนคี่

วัตถุประสงค์หลัก : ตรวจสอบตัวเลขจำนวนเต็มว่าเป็นจำนวนคู่ หรือจำนวนคี่

ข้อมูลนำเข้า : เลขจำนวนเต็ม 1 จำนวน

ตัวแปรที่ใช้ : num แทน เลขจำนวนเต็มหนึ่งจำนวน

การประมวลผล : ถ้า num mod 2 = 0 พิมพ์ “Even number”

ถ้า num mod 2 ≠ 0 พิมพ์ “Odd number”

ข้อมูลส่งออก (ผลลัพธ์) : “Even number”

“Odd number”

ขั้นตอนวิธี

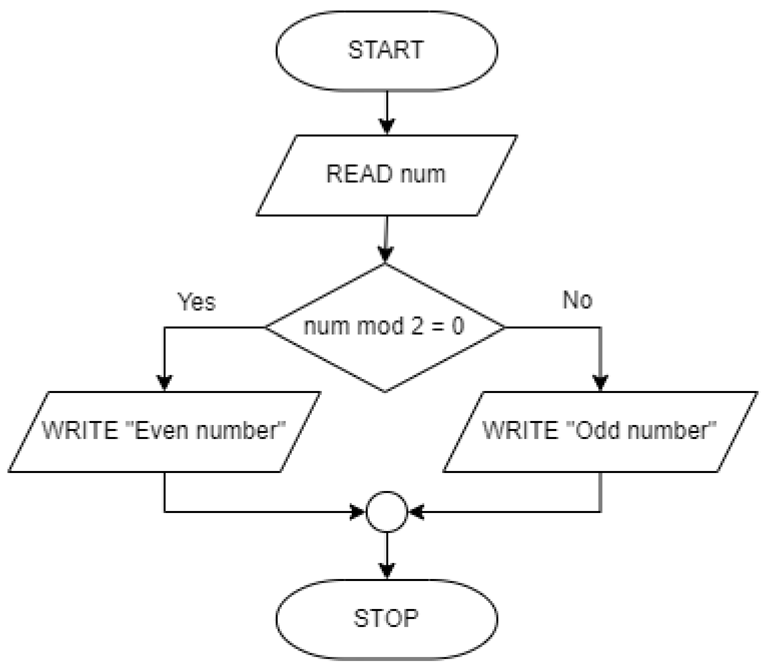
1. เริ่มต้น
2. รับข้อมูลเข้า แล้วนำมาเก็บไว้ในตัวแปร num
3. ถ้าหากค่าในตัวแปร num mod 2 = 0 แล้ว

   3.1 แสดงข้อความ “Even number”

   ไม่เช่นนั้น

   3.2 แสดงข้อความ “Odd number”
4. จบการทำงาน

ผังงาน



ตัวอย่างที่ 4 รับตัวเลขจำนวนเต็ม 2 จำนวน แล้วเปรียบเทียบว่าจำนวนไหนมากกว่ากัน ถ้าเท่ากันให้แสดงว่าเท่ากัน

วัตถุประสงค์หลัก : เปรียบเทียบเลขจำนวนเต็ม 2 จำนวน

ข้อมูลนำเข้า : เลขจำนวนเต็ม 2 จำนวน

ตัวแปรที่ใช้ : num1 แทน เลขจำนวนที่ 1
num2 แทน เลขจำนวนที่ 2

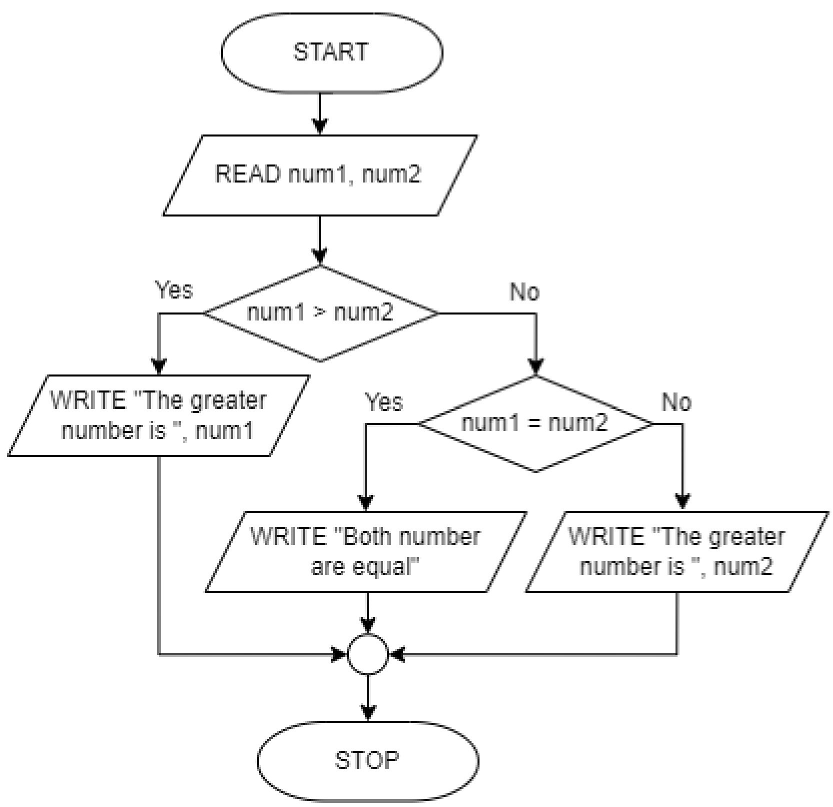
การประมวลผล : ถ้า num1 > num2 พิมพ์ "The greater number is ", num1
ถ้า num1 < num2 พิมพ์ "The greater number is ", num2
ถ้า num1 = num2 พิมพ์ "Both number are equal"

ข้อมูลส่งออก (ผลลัพธ์) : "The greater number is ", เลขจำนวนที่มากกว่า
"Both number are equal"

ขั้นตอนวิธี

1. เริ่มต้น
2. รับข้อมูลเข้า แล้วนำมาเก็บไว้ใน ตัวแปร num1 และ num2
3. ถ้าหากค่าในตัวแปร num1 > num2 แล้ว
   3.1 แสดง "The greater number is ", num1
   3.2 ไม่เช่นนั้น ถ้าหากค่าในตัวแปร num1 = num2 แล้ว
      3.2.1 แสดงข้อความ "Both number are equal" ไม่เช่นนั้น
      3.2.2 แสดง "The greater number is ", num2
4. จบการทำงาน

ผังงาน



<u>ตัวอย่างที่ 5</u> หาชั้นปีของนักศึกษา

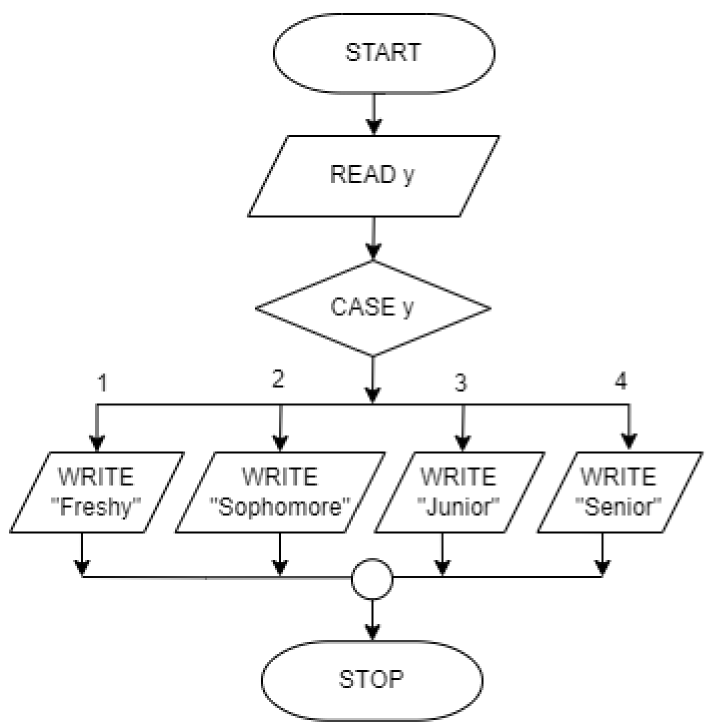
"Freshy" หมายถึง ชั้นปีที่ 1
"Sophomore" หมายถึง ชั้นปีที่ 2
"Junior" หมายถึง ชั้นปีที่ 3
"Senior" หมายถึง ชั้นปีที่ 4

วัตถุประสงค์หลัก : ชั้นปีของนักศึกษา
ข้อมูลนำเข้า : ตัวเลขปีของนักศึกษา
ตัวแปรที่ใช้ : y แทน ปีของนักศึกษา
การประมวลผล : CASE y
1 พิมพ์ "Freshy"
2 พิมพ์ "Sophomore"
3 พิมพ์ "Junior"
4 พิมพ์ "Senior"

ขั้นตอนวิธี

1. เริ่มต้น
2. รับข้อมูลเข้า แล้วนำมาเก็บไว้ในตัวแปร y
3. ตรวจสอบค่าตัวแปร y ตามเงื่อนไข
   3.1 ถ้าตัวเลขมีค่า 1 ให้แสดงข้อความ "Freshy"
   3.2 ถ้าตัวเลขมีค่า 2 ให้แสดงข้อความ "Sophomore"
   3.3 ถ้าตัวเลขมีค่า 3 ให้แสดงข้อความ "Junior"
   3.4 ถ้าตัวเลขมีค่า 4 ให้แสดงข้อความ "Senior"
4. จบการทำงาน

ผังงาน



<u>ตัวอย่างที่ 6</u> หาผลบวกของเลขตั้งแต่ 1 ถึง 10

วัตถุประสงค์หลัก : หาผลบวกของเลขตั้งแต่ 1 ถึง 10

ข้อมูลนำเข้า : -

ตัวแปรที่ใช้ : sum แทน ผลบวก

a แทน ตัวเลข

การประมวลผล : sum = 0 และ a = 1

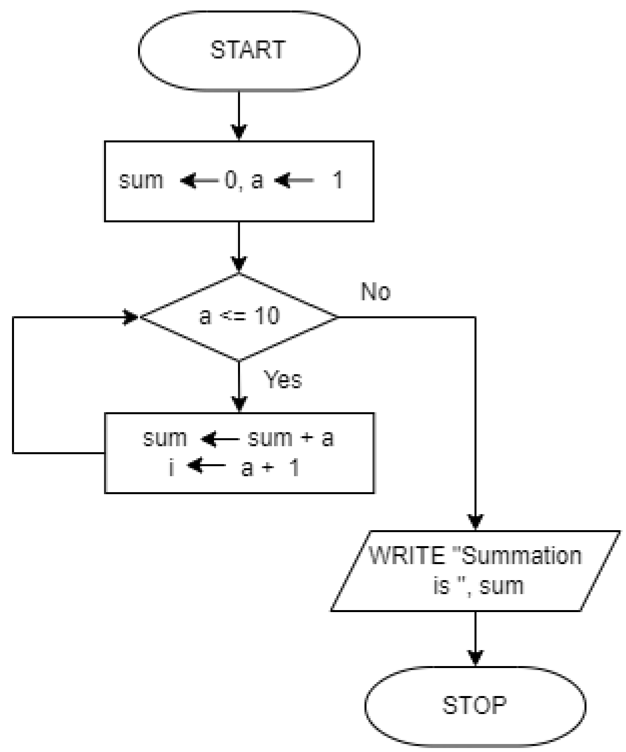
ในขณะที่ ค่าของ a ≤ 10 จะทำ sum = sum + a และ a = a + 1

ข้อมูลส่งออก (ผลลัพธ์) : “Summation is “

ขั้นตอนวิธี

1. เริ่มต้น
2. กำหนดให้ sum = 0 และ a = 1
3. ในขณะที่ ค่าของ a ≤ 10
   3.1 นำค่า sum + a ไปเก็บไว้ในตัวแปร Sum
   3.2 นำค่า a + 1 ไปเก็บไว้ในตัวแปร a
4. แสดงข้อความ “Summation is ”
5. แสดงค่าในตัวแปร sum
6. จบการทำงาน

ผังงาน



<u>ตัวอย่างที่ 7</u> หาผลบวกของเลขคู่ตั้งแต่ 2 ถึง 100

วัตถุประสงค์หลัก :	หาผลบวกของเลขคู่ตั้งแต่ 2 ถึง 100
ข้อมูลนำเข้า :	-
ตัวแปรที่ใช้ :	sum แทน ผลบวก
	a แทน เลขคู่
การประมวลผล :	sum = 0 และ a = 2
	ทำซ้ำ sum = sum + i และ a = a + 2 จนกระทั่ง a > 100
ข้อมูลส่งออก (ผลลัพธ์) :	"Summation is "

ขั้นตอนวิธี

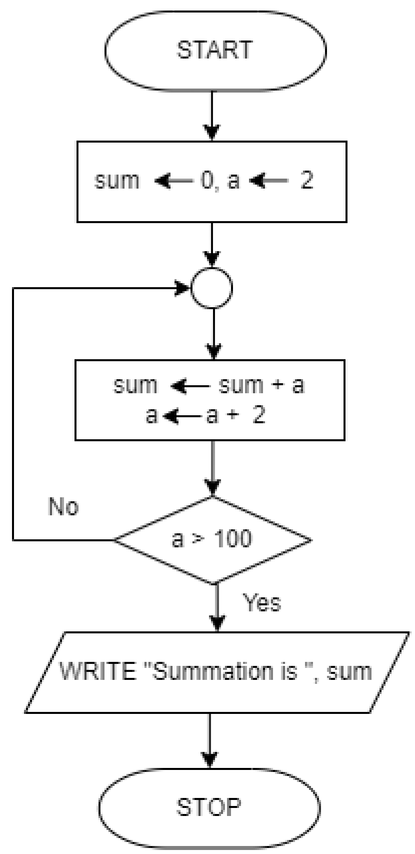
1. เริ่มต้น
2. กำหนดให้ sum = 0 และ a = 2
3. ทำซ้ำ

   3.1 นำค่า sum + a ไปเก็บไว้ในตัวแปร Sum

   3.2 นำค่า a + 2 ไปเก็บไว้ในตัวแปร a

   จนกระทั่ง a > 100
4. แสดงข้อความ "Summation is "
5. แสดงค่าในตัวแปร sum
6. จบการทำงาน

ผังงาน



<u>ตัวอย่างที่ 8</u> รับตัวเลขจำนวนเต็ม 10 จำนวน แล้วทำการหาค่าเฉลี่ย

วัตถุประสงค์หลัก : การหาค่าเฉลี่ย

ข้อมูลนำเข้า : เลขจำนวนเต็ม 10 จำนวน

ตัวแปรที่ใช้ : num แทน ตัวเลขจำนวนเต็มใด ๆ ทั้งหมด 10 จำนวน

sum แทน ผลรวม

a แทน จำนวนนับ

การประมวลผล : กำหนดให้ sum = 0

ทำซ้ำ FOR a = 1 TO 10 ทำ sum = sum + num

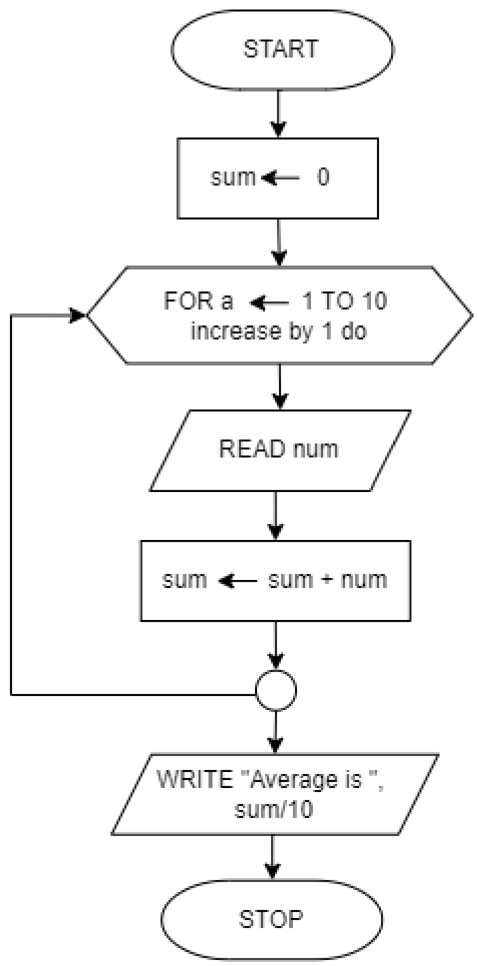
แสดงค่า sum/10

ข้อมูลส่งออก (ผลลัพธ์) : “Average is “

ขั้นตอนวิธี

1. เริ่มต้น
2. กำหนดให้ sum = 0
3. ทำซ้ำ FOR a = 1 TO 10
   3.1 รับข้อมูลเข้า แล้วนำมา เก็บไว้ในตัวแปร num
   3.2 นำค่า sum + num ไป เก็บไว้ในตัวแปร sum
   จนกระทั่ง a > 10
4. แสดงข้อความ “Average is ”
5. แสดงค่าในตัวแปร sum/10
6. จบการทำงาน

ผังงาน

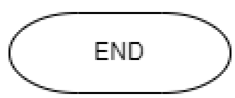
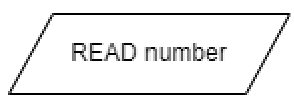
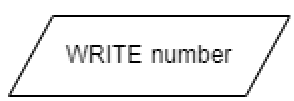
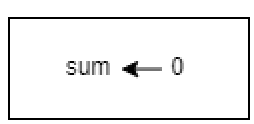
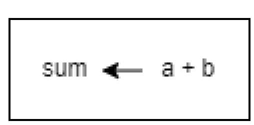


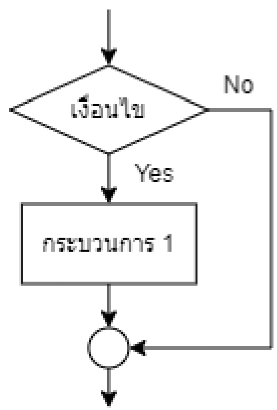
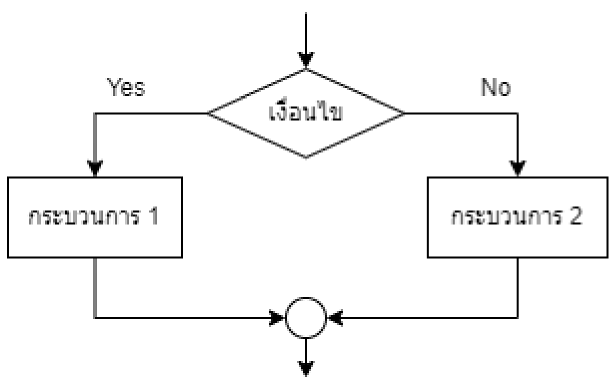
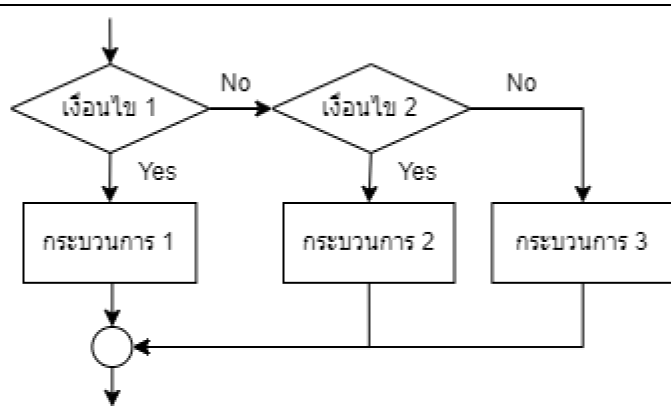
3.4 การเขียนขั้นตอนวิธีแบบรหัสเทียม (Pseudo Code)

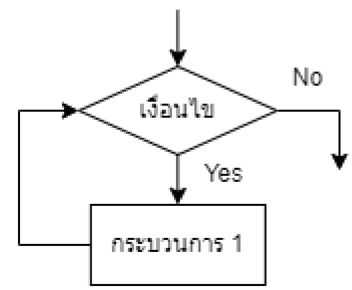
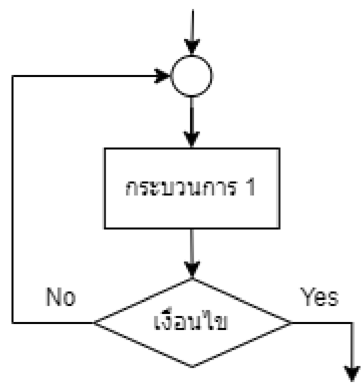
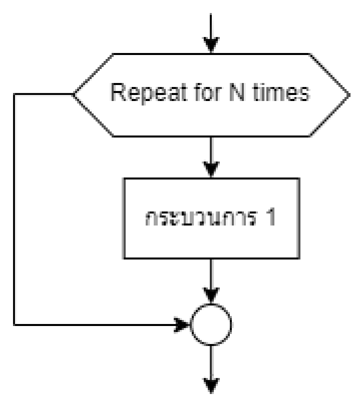
รหัสเทียม คือ การเขียนคำอธิบายการทำงานของขั้นตอนวิธี (Algorithm) โดยใช้ประโยคภาษาอังกฤษที่สื่อความหมายง่าย ๆ สามารถอ่านแล้วเข้าใจได้โดยทันที โดยที่ไม่คำนึงไวยากรณ์ และไม่ขึ้นอยู่กับภาษาโปรแกรมใด ๆ

หลักการเขียนรหัสเทียม

1. ใช้ถ้อยคำที่เข้าใจง่าย
2. ในหนึ่งบรรทัด มีเพียงหนึ่งการกระทำ
3. เขียนคำสั่งการทำงานจากบนลงล่าง
4. ใช้ย่อหน้า เพื่อแสดงกลุ่มของคำสั่งที่ทำงานในระดับเดียวกัน

ขั้นตอนวิธี	ผังงาน	รหัสเทียม
เริ่มต้นการทำงาน		START, BEGIN
สิ้นสุดการทำงาน		STOP, END
รับข้อมูลเข้า		READ number, GET number, INPUT number
แสดงผลข้อมูลออก		WRITE number, OUTPUT number
การกำหนดค่า		SET sum ← 0
การคำนวณ		COMPUTE sum ← a + b CALCULATE sum ← a + b

<table>
<tr><th>ขั้นตอนวิธี</th><th>ผังงาน</th><th>รหัสเทียม</th></tr>
<tr><td>ทางเลือก</td><td>
</td><td>IF เงื่อนไข THEN
   กระบวนการ 1
END IF</td></tr>
<tr><td></td><td>
</td><td>IF เงื่อนไข THEN
   กระบวนการ 1
ELSE
   กระบวนการ 2
END IF</td></tr>
<tr><td></td><td>
</td><td>IF เงื่อนไข 1 THEN
   กระบวนการ 1
ELSE
   IF เงื่อนไข 2 THEN
      กระบวนการ 2
   ELSE
      กระบวนการ 3
END IF</td></tr>
</table>

ขั้นตอนวิธี	ผังงาน	รหัสเทียม
การทำซ้ำในขณะที่		WHILE เงื่อนไข DO กระบวนการ 1 END WHILE
การทำซ้ำจนกระทั่ง		REPEAT กระบวนการ 1 UNTIL เงื่อนไข
การทำซ้ำแบบ For		FOR i = start TO stop increase by N times DO กระบวนการ 1 END FOR หรือ FOR i = stop TO start decrease by N times DO กระบวนการ 1 END FOR

3.3 ตัวอย่างการประยุกต์ใช้งานรหัสเทียม

ตัวอย่างที่ 9 หาพื้นที่วงกลมและแสดงผลลัพธ์

ข้อมูลนำเข้า : รัศมีของวงกลม

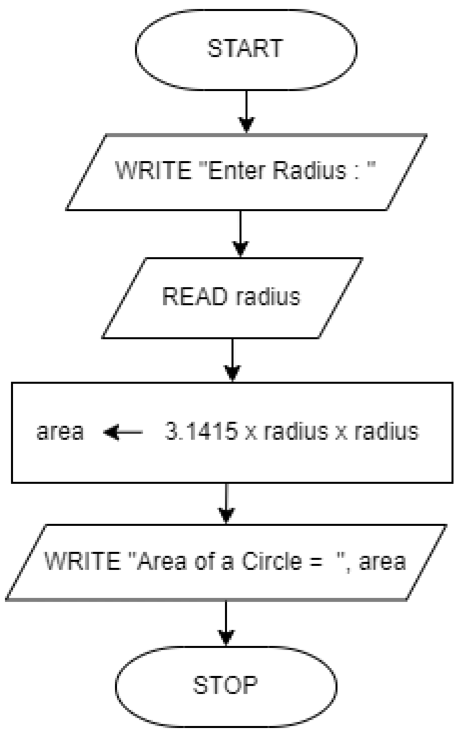
ตัวแปรที่ใช้ : Radius แทน รัศมีของวงกลม

Area แทน พื้นที่วงกลม

การประมวลผล : Area คำนวณได้จากสูตร 3.1415 x รัศมี x รัศมี

ข้อมูลส่งออก (ผลลัพธ์) : พื้นที่วงกลม

ผังงาน



รหัสเทียม

```
BEGIN
WRITE "Enter Radius : "
READ radius
COMPUTE
    area ← 3.1415 x radius x radius
WRITE "Area of a Circle = ", area
END
```

<u>ตัวอย่างที่ 10</u> หาพื้นที่สามเหลี่ยมและแสดงผลลัพธ์

ข้อมูลนำเข้า : ความยาวฐาน และความสูง

ตัวแปรที่ใช้ : Base แทน ความยาวฐาน

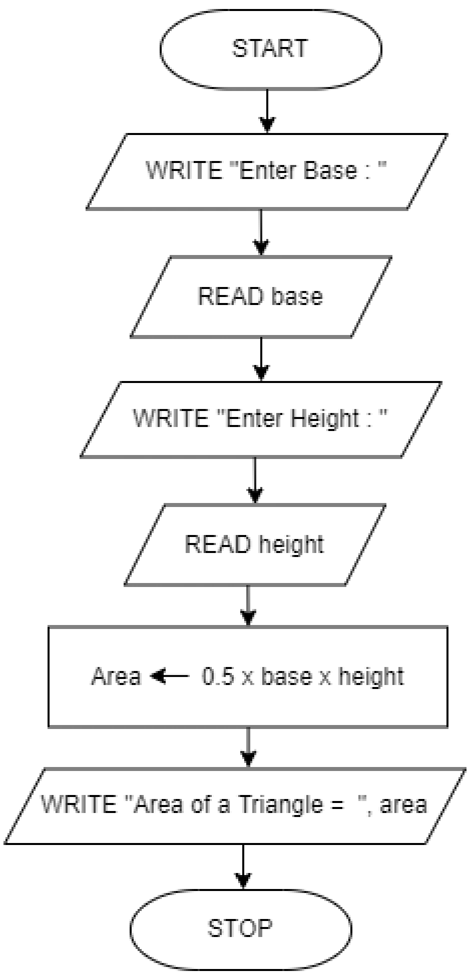
Height แทน ความสูง

Area แทน พื้นที่สามเหลี่ยม

การประมวลผล : Area คำนวณได้จากสูตร 0.5 x ความยาวฐาน x ความสูง

ข้อมูลส่งออก (ผลลัพธ์) : พื้นที่สามเหลี่ยม

ผังงาน



รหัสเทียม

BEGIN

WRITE “Enter Base : “

READ base

WRITE “Enter Height : “

READ height

COMPUTE

area ← 0.5 x Base x Height

WRITE “Area of a Triangle = “, area

END

<u>ตัวอย่างที่ 11</u> รับเลขจำนวนเต็มมาหนึ่งจำนวน และแสดงข้อความนั้นเป็นจำนวนคู่ หรือจำนวนคี่

วัตถุประสงค์หลัก : ตรวจสอบตัวเลขจำนวนเต็มว่าเป็นจำนวนคู่ หรือจำนวนคี่

ข้อมูลนำเข้า : เลขจำนวนเต็ม 1 จำนวน

ตัวแปรที่ใช้ : num แทน เลขจำนวนเต็มหนึ่งจำนวน

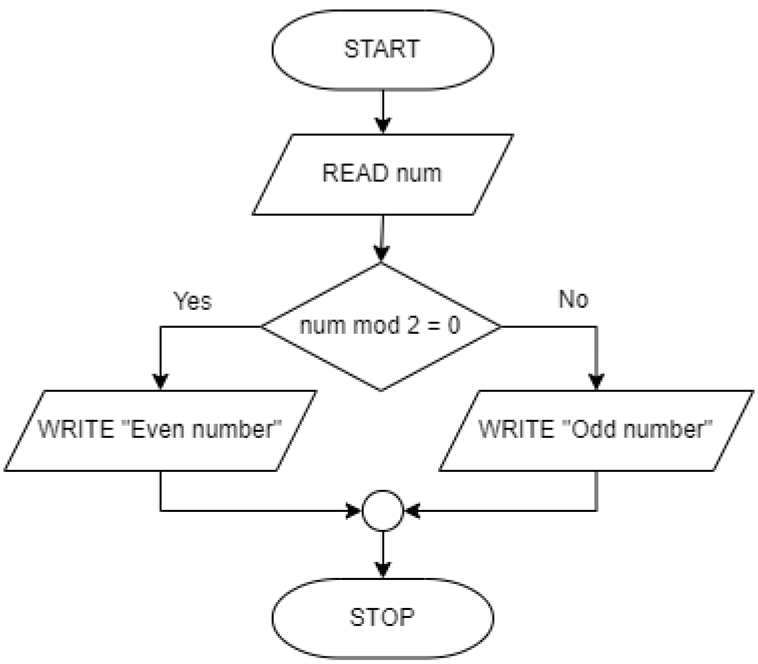
การประมวลผล : ถ้า num mod 2 = 0 พิมพ์ "Even number"

ถ้า num mod 2 ≠ 0 พิมพ์ "Odd number"

ข้อมูลส่งออก (ผลลัพธ์) : "Even number"

"Odd number"

ผังงาน



รหัสเทียม

```
BEGIN
READ num
IF num mod 2 = 0 THEN
    WRITE "Even number "
ELSE
    WRITE "Odd number"
END IF
END
```

ตัวอย่างที่ 12 รับตัวเลขจำนวนเต็ม 2 จำนวน และเปรียบเทียบว่าจำนวนไหนมากกว่ากัน ถ้าเท่ากันให้แสดงว่าเท่ากัน

วัตถุประสงค์หลัก : เปรียบเทียบเลขจำนวนเต็ม 2 จำนวน

ข้อมูลนำเข้า : เลขจำนวนเต็ม 2 จำนวน

ตัวแปรที่ใช้ : num1 แทน เลขจำนวนที่ 1

num2 แทน เลขจำนวนที่ 2

การประมวลผล : ถ้า num1 > num2 พิมพ์ "The greater number is ", num1

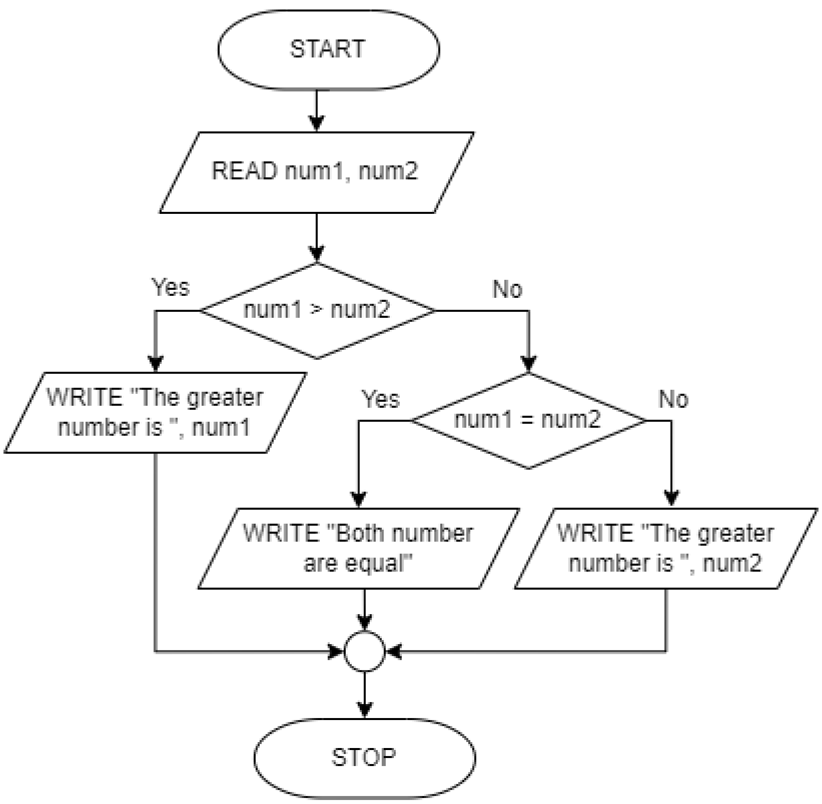
ถ้า num1 < num2 พิมพ์ "The greater number is ", num2

ถ้า num1 = num2 พิมพ์ "Both number are equal"

ข้อมูลส่งออก (ผลลัพธ์) : "The greater number is ", เลขจำนวนที่มากกว่า

"Both number are equal"

ผังงาน



รหัสเทียม

BEGIN

READ num1, num2

IF num1 > num2 THEN

WRITE "The greater number is ", num1

ELSE

IF num1 = num2 THEN

WRITE "Both number are equal"

ELSE

WRITE "The greater number is ", num2

END IF

END

<u>ตัวอย่างที่ 13</u> แสดงเกรดของนศ.คนหนึ่งจากคะแนนเต็ม 100

คะแนน 80 - 100	ได้เกรด A
คะแนน 70 - 79	ได้เกรด B
คะแนน 60 - 69	ได้เกรด C
คะแนน 50 – 59	ได้เกรด D
คะแนน 0 - 49	ได้เกรด F

ข้อมูลนำเข้า : เลขจำนวนเต็ม 1 จำนวน

ตัวแปรที่ใช้ : score แทน คะแนน

การประมวลผล : ถ้า score > 79 พิมพ์ “A”

ถ้า score > 69 พิมพ์ “B”

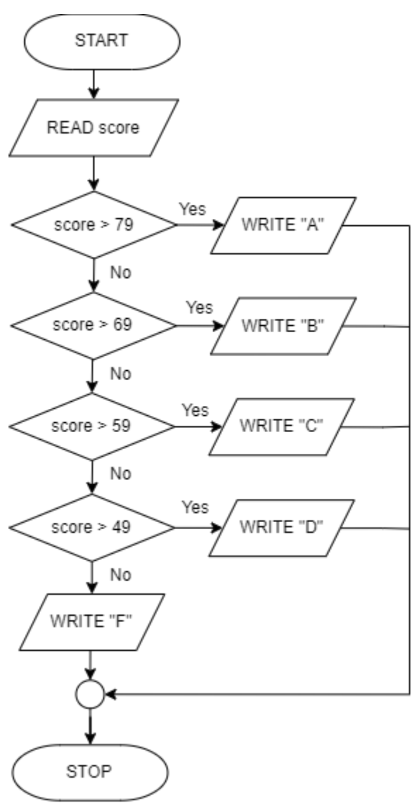
ถ้า score > 59 พิมพ์ “C”

ถ้า score > 49 พิมพ์ “D”

ถ้า score < 50 พิมพ์ “F”

ข้อมูลส่งออก (ผลลัพธ์) : เกรด

ผังงาน



รหัสเทียม

```
BEGIN
READ score
IF score > 79 THEN
    WRITE “A”
ELSE IF score > 69 THEN
    WRITE “B”
ELSE IF score > 59 THEN
    WRITE “C”
ELSE IF score > 49 THEN
    WRITE “D”
ELSE
    WRITE “F”
END IF
END
```

<u>ตัวอย่างที่ 14</u> หาชั้นปีของนักศึกษา

“Freshy”	หมายถึง ชั้นปีที่ 1
“Sophomore”	หมายถึง ชั้นปีที่ 2
“Junior”	หมายถึง ชั้นปีที่ 3
“Senior”	หมายถึง ชั้นปีที่ 4
“Invalid”	หมายถึง อื่น ๆ

วัตถุประสงค์หลัก : ชั้นปีของนักศึกษา

ข้อมูลนำเข้า : ตัวเลขปีของนักศึกษา

ตัวแปรที่ใช้ : y แทน ปีของนักศึกษา

การประมวลผล : CASE y

1 พิมพ์ “Freshy”

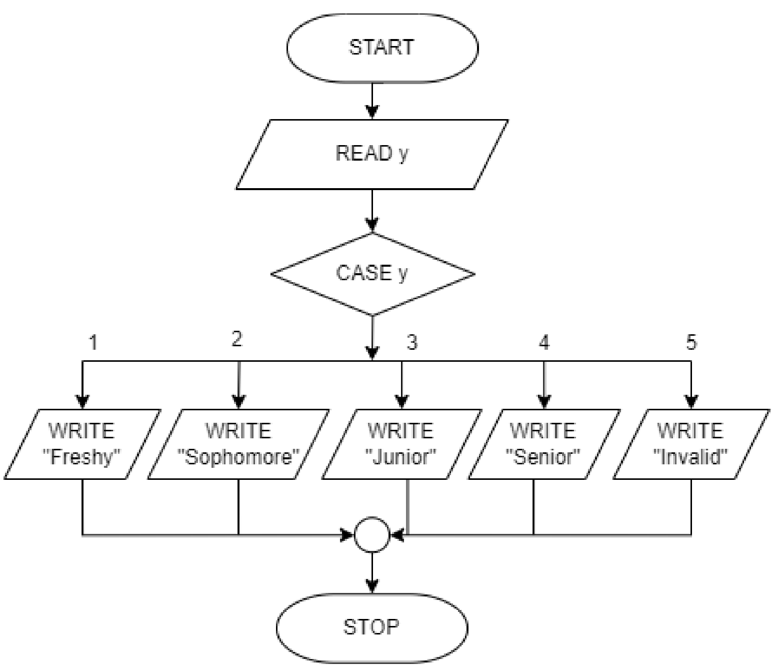
2 พิมพ์ “Sophomore”

3 พิมพ์ “Junior”

4 พิมพ์ “Senior”

อื่น ๆ พิมพ์ “Invalid”

ผังงาน



รหัสเทียม

```
BEGIN
READ y
CASE y
    CASE “1” : WRITE “Freshy”
    CASE “2” :
        WRITE “Sophomore”
    CASE “3” :
        WRITE “Junior”
    CASE “4” :
        WRITE “Senior”
    DEFAULT :
        WRITE “Invalid”
END CASE
```

END

<u>ตัวอย่างที่ 15</u> หาผลบวกของเลขตั้งแต่ 1 ถึง 10

วัตถุประสงค์หลัก : หาผลบวกของเลขตั้งแต่ 1 ถึง 10

ข้อมูลนำเข้า : -

ตัวแปรที่ใช้ : sum แทน ผลบวก

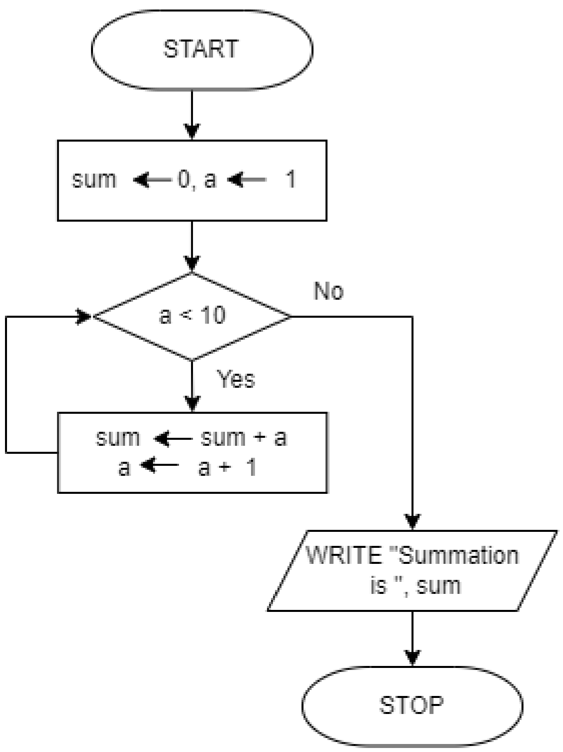
a แทน ตัวเลข

การประมวลผล : sum = 0 และ a = 1

ในขณะที่ ค่าของ a < 10 จะทำ sum = sum + a และ a = a + 1

ข้อมูลส่งออก (ผลลัพธ์) : "Summation is "

ผังงาน



รหัสเทียม

```
BEGIN
SET sum ← 0, a ← 1
WHILE a < 10 DO
    COMPUTE
        Sum ← Sum + a
    COMPUTE
        a ← a + 1
END WHILE
WRITE "Summation is ", sum
END
```

<u>ตัวอย่างที่ 16</u> หาผลบวกของเลขคู่ตั้งแต่ 2 ถึง 100

วัตถุประสงค์หลัก : หาผลบวกของเลขคู่ตั้งแต่ 2 ถึง 100

ข้อมูลนำเข้า : -

ตัวแปรที่ใช้ : sum แทน ผลบวก

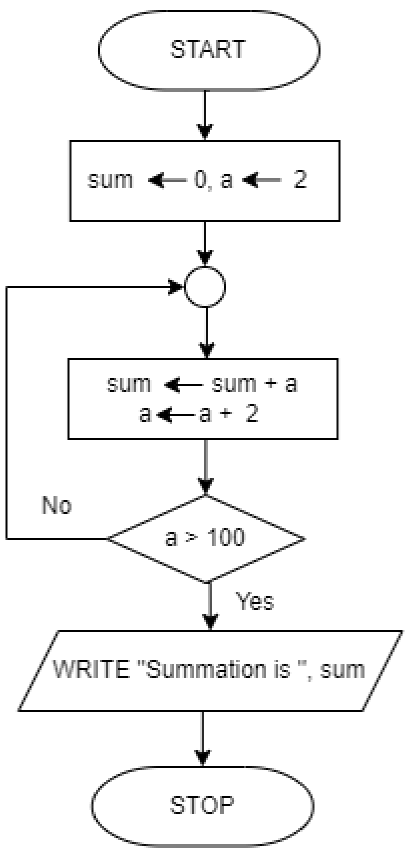
a แทน เลขคู่

การประมวลผล : sum = 0 และ a = 2

ทำซ้ำ sum = sum + i และ a = a+ 2 จนกระทั่ง a > 100

ข้อมูลส่งออก (ผลลัพธ์) : “Summation is “

ผังงาน



รหัสเทียม

```
BEGIN
SET sum ← 0, a ← 2
REPEAT
    COMPUTE
        Sum ← Sum + a
    COMPUTE
        a ← a + 2
UNTIL a > 100
WRITE “Summation is “, sum
END
```

ตัวอย่างที่ 17 รับตัวเลขจำนวนเต็ม 10 จำนวน แล้วทำการหาค่าเฉลี่ย

วัตถุประสงค์หลัก : การหาค่าเฉลี่ย

ข้อมูลนำเข้า : เลขจำนวนเต็ม 10 จำนวน

ตัวแปรที่ใช้ : num แทน ตัวเลขจำนวนเต็มใด ๆ ทั้งหมด 10 จำนวน

sum แทน ผลรวม

a แทน จำนวนนับ

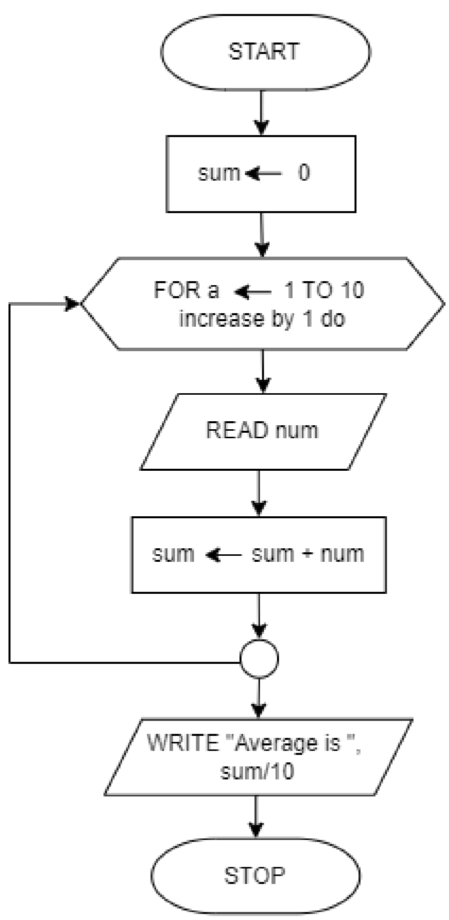
การประมวลผล : กำหนดให้ sum = 0

ทำซ้ำ FOR a = 1 TO 10 ทำ sum = sum + Num

แสดงค่า sum/10

ข้อมูลส่งออก (ผลลัพธ์) : “Average is “

ผังงาน



รหัสเทียม

BEGIN

SET sum ← 0

For a = 1 TO 10 increase by 1 do

READ num

COMPUTE

sum ← sum + num

END For

WRITE “Average is ”, sum/10

END

บทที่ 4
ชนิดข้อมูลและตัวแปร (Data Type And Variable)

บทนี้กล่าวถึงเรื่อง (1) ชนิดข้อมูลและตัวแปร เนื่องจากสิ่งที่ต้องศึกษาเป็นลำดับแรก ในการเขียนโปรแกรมภาษาซีคือ การทำความเข้าใจพื้นฐานชนิดของข้อมูลและ หลักการของการใช้งานตัวแปร จากนั้นกล่าวถึง (2) ชนิดของข้อมูล เนื่องจากภาษาซี มีข้อมูลให้เลือกใช้หลายชนิด การเข้าใจพื้นฐานชนิดของข้อมูลจะส่งผลให้ สามารถเลือกใช้งานข้อมูลที่เหมาะสมในการเขียนโปรแกรม ซึ่งจะทำให้ได้โปรแกรมต้นฉบับ ที่มีประสิทธิภาพและเป็นการใช้ทรัพยากรของเครื่องคอมพิวเตอร์ได้อย่างคุ้มค่า และสุดท้ายกล่าวถึง (3) การแปลงชนิดของข้อมูลเพื่อจะสามารถใช้งานตัวแปรแบบต่างชนิดเพื่อร่วมกันทำงานในการเขียนโปรแกรมได้อย่างมีประสิทธิภาพ

4.1 ตัวแปร

ในมุมมองของโปรแกรมเมอร์ ตัวแปรคือสัญลักษณ์ที่กำหนดขึ้นเพื่อใช้แทนค่าของข้อมูล ในการเขียนโปรแกรม ซึ่งข้อมูลที่เก็บในตัวแปรนั้นจะเป็นตัวหนังสือหรือตัวเลขก็ได้ แต่ในมุมมองของ การทำงานของคอมพิวเตอร์ตัวแปรคือชื่อที่ใช้แทนตำแหน่งของหน่วยความจำที่สามารถเปลี่ยนแปลงค่าได้ภายในขอบเขตของชนิดข้อมูลที่กำหนด การใช้งานตัวแปรต้องประกาศชนิดของข้อมูล และตั้งชื่อตัวแปรก่อนที่จะใช้งานเสมอ การประกาศตัวแปรเป็นการกำหนดว่าตัวแปรนั้นจะเก็บข้อมูลชนิดใด โดยที่ตัวแปรจะไม่สามารถเก็บค่าชนิดข้อมูลที่แตกต่างจากที่กำหนดไว้

ในมุมมองด้านการทำงานของฮาร์ดแวร์ การประกาศตัวแปรคือการจองพื้นที่ของหน่วยความจำ และการกำหนดค่าให้กับตัวแปรคือการนำข้อมูลไปเก็บไว้ในพื้นที่ของหน่วยความจำ ที่จองไว้ ดังรูปที่ 4.1

การประกาศตัวแปร

ตัวแปร	หน่วยความจำ
counter	
price	
alphabet	
message	
GPA	

การกำหนดค่าให้ตัวแปร

ตัวแปร	หน่วยความจำ
counter	5
price	200
alphabet	A
message	C Programming
GPA	3.95

รูปที่ 4.1 การประกาศและกำหนดค่าให้กับตัวแปร

4.1.1 กฎการตั้งชื่อตัวแปร

การตั้งชื่อตัวแปรที่ดีนั้นมีความสำคัญในการเขียนโปรแกรม ชื่อตัวแปรที่ดีช่วยสื่อถึงหน้าที่ ของตัวแปร ชื่อตัวแปรที่ดีจะต้องไม่สั้นและยาวจนเกินไป ชื่อตัวแปรที่สั้นจะไม่สื่อความหมาย ถึงหน้าที่ของตัวแปร ส่วนชื่อตัวแปรที่ยาวถึงแม้จะสื่อถึงหน้าที่ของตัวแปรได้ดีกว่าแต่อาจจะทำให้โปรแกรม อ่านยากกว่าชื่อตัวแปรที่สั้น ในการเขียนโปรแกรมภาษาซีมีกฎในการตั้งชื่อตัวแปร 5 ข้อดังนี้

1) **ต้องขึ้นต้นด้วยตัวอักษรหรือเครื่องหมาย** underscore (_) เท่านั้น โดยไม่สามารถขึ้นต้นด้วยตัวเลขหรือเครื่องหมายอื่น แต่ภายในชื่อสามารถประกอบด้วยตัวอักษร ตัวเลข หรือเครื่องหมาย underscore ก็ได้
2) **ห้ามใช้อักขระพิเศษ**ในชื่อของตัวแปร เช่น # $ @ & เป็นต้น
3) **ห้ามมีช่องว่าง**หรือแท็บภายในชื่อของตัวแปร
4) **ชื่อตัวแปรเป็นแบบ case sensitive** คือตัวอักษรตัวเล็กและตัวอักษรตัวใหญ่ถือว่าเป็นคนละตัวเช่น การตั้งชื่อตัวแปร 3 ตัวว่า Program program และ pRoGrAm ถือว่าเป็นตัวแปรคนละตัว
5) **ต้องไม่ซ้ำกับคำสงวน** (reserve word) จำนวน 32 คำ ดังรูปที่ 4.2

auto	break	case	char	const
continue	default	do	double	else
enum	extern	float	for	goto
if	int	long	register	return
short	signed	sizeof	static	struct
switch	typedef	union	unsigned	void
volatile	while			

รูปที่ 4.2 คำสงวนในภาษาซี

4.1.2 การประกาศตัวแปร

การประกาศตัวแปรคือการจองพื้นที่ของหน่วยจำเพื่อใช้ในการเก็บข้อมูล ขนาดของหน่วยความจำที่ทำการจองขึ้นอยู่กับประเภทและชนิดของตัวแปรที่ทำการประกาศ การประกาศตัวแปรสามารถทำได้ทุกส่วนของโปรแกรมซึ่งจะส่งผลให้คุณสมบัติในการเข้าถึงตัวแปรแตกต่าง กันไปตามตำแหน่งที่ทำการประกาศ

ตัวอย่างที่ 4.1 การประกาศตัวแปร

```
1  #include<stdio.h>
2
3  int number;
4  char message;
5
6  void function1()
7  {
8       float GPA;
9       char string_test[20] ;
10 }
11
12 void main()
13 {
14      int counter;
15 }
```

คำอธิบายตัวอย่างที่ 4.1

บรรทัดที่ 1	เรียกใช้งานส่วนของเฮดเดอร์ไฟล์ ไลบรารี stdio.h ซึ่งทำหน้าที่จัดการอินพุต และเอาต์พุต
บรรทัดที่ 3	ประกาศตัวแปรชนิดจำนวนเต็มชื่อ number ภายนอกฟังก์ชัน
บรรทัดที่ 4	ประกาศตัวแปรชนิดตัวอักษรชื่อ message ภายนอกฟังก์ชัน
บรรทัดที่ 6	การประกาศฟังก์ชันแบบกำหนดเองชื่อ function1()
บรรทัดที่ 8	ประกาศตัวแปรชนิดจำนวนทศนิยมชื่อ GPA ภายในฟังก์ชัน
บรรทัดที่ 9	ประกาศตัวแปรชนิดข้อความชื่อ string_test (Array) ภายในฟังก์ชัน
บรรทัดที่ 12	การประกาศฟังก์ชันหลัก
บรรทัดที่ 14	ประกาศตัวแปรชนิดจำนวนเต็มชื่อ counter ภายในฟังก์ชัน

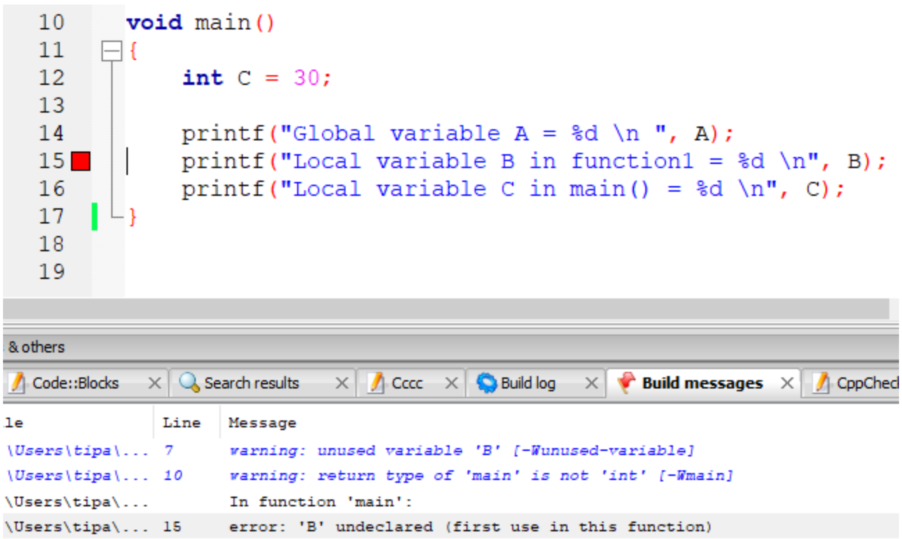
4.1.3 ประเภทของตัวแปร

ตัวแปรในภาษาซีสามารถแบ่งได้ 2 ประเภท ประกอบด้วยตัวแปรแบบโลคอล และตัวแปรแบบโกลบอล **ตัวแปรแบบโลคอล** คือตัวแปรที่ประกาศไว้ภายในฟังก์ชันใดฟังก์ชันหนึ่ง ส่งผลให้ฟังก์ชันนั้น เป็นเจ้าของตัวแปร ที่ประกาศไป ฟังก์ชันอื่นไม่สามารถเรียกใช้งานตัวแปรนั้นได้ **ตัวแปรแบบโกลบอล** คือตัวแปรที่ประกาศไว้ภายนอกฟังก์ชันส่งผลให้สามารถเรียกใช้งานตัวแปรนั้นได้จากทุกส่วนในโปรแกรม ต้นฉบับพิจารณาตัวอย่างที่ 4.2 เพื่อความเข้าใจเรื่องตัวแปรแบบโลคอลและโกลบอล

ตัวอย่างที่ 4.2 ตัวแปรแบบโลคอลและโกลบอล

```
1   #include<stdio.h>
2
3   int A = 10;
4
5   void function1()
6   {
7       int B = 20;
8   }
9
10  void main()
11  {
12      int C = 30;
13
14      printf("Global variable A = %d \n ", A);
15      printf("Local variable B in function1 = %d \n", B);
16      printf("Local variable C in main() = %d \n", C);
17  }
```

ผลลัพธ์ตัวอย่างที่ 4.2



รูปที่ 4.3 ผลลัพธ์ตัวอย่างที่ 4.2

จะพบว่าคอมไพเลอร์แสดงข้อผิดพลาดว่าไม่รู้จักตัวแปร B เนื่องจากเป็นตัวแปร แบบโลคอลที่ประกาศในฟังกชัน function1() จึงไม่สามารถเรียกใช้งานตัวแปร B ในฟังก์ชัน main() ได้ ถ้าหากต้องการให้โปรแกรมต้นฉบับนี้ทำงานได้ ต้องแก้ไขโปรแกรมบรรทัดที่ 15 โดยการเรียกใช้ตัวแปรที่ประกาศไว้ในฟังก์ชัน เดียวกันกับคำสั่ง หรือเรียกใช้ตัวแปรโกลบอล

คำอธิบายตัวอย่างที่ 4.2

บรรทัดที่ 1 เรียกใช้งานส่วนของเฮดเดอร์ไฟล์ ไลบรารี stdio.h ซึ่งทำหน้าที่จัดการอินพุต และเอาต์พุต

บรรทัดที่ 3 ประกาศตัวแปรชนิดจำนวนเต็มประเภทตัวแปรโกลบอลชื่อ A และกำหนดค่าเป็น 10

บรรทัดที่ 5 การประกาศฟังก์ชันกำหนดเองชื่อ function1()

บรรทัดที่ 7 ประกาศตัวแปรชนิดจำนวนเต็มประเภทตัวแปรโลคอลชื่อ B และกำหนดค่าเป็น 20

บรรทัดที่ 14 ถึง 16 แสดงผลค่าของตัวแปร 1) A 2) B และ 3) C ตามลำดับ

4.2 ข้อมูล

การเขียนโปรแกรมภาษาซีมีประเภทของข้อมูลให้เลือกใช้อย่างหลากหลาย ข้อมูลแต่ละประเภท มีความเหมาะสมในการนำไปใช้งานแตกต่างกันข้อมูลที่มีขอบเขตมากก็มีความต้องการพื้นที่ของหน่วยความจำมาก เช่นกัน ในการเขียนโปรแกรมภาษาซี ควรคำนึงถึงความจำเป็นใน การเลือกใช้งาน ข้อมูลแต่ละชนิดให้เหมาะสมกับลักษณะของงาน หากเลือกใช้งานชนิดของข้อมูลที่มีขนาดน้อยกว่าขอบเขต ของข้อมูลที่ต้องการใช้งานจะส่งผลให้เกิดข้อผิดพลาดในการทำงานของโปรแกรม แต่หากเลือกใช้ชนิดข้อมูล ที่มีขอบเขตใหญ่เกินความจำเป็น จะส่งผลให้สิ้นเปลืองพื้นที่ของหน่วยความจำ การเริ่มศึกษาเรื่องประเภท ของข้อมูล นิยมแบ่งประเภทของข้อมูลเป็น 5 ประเภท ประกอบด้วย ข้อมูลแบบจำนวนเต็ม ข้อมูลแบบ จำนวนทศนิยม ข้อมูลแบบตัวอักษร ข้อมูลแบบข้อความ และข้อมูลแบบค่าคงที่ โดยข้อมูลแต่ละประเภท มีรายละเอียดดังนี้

4.2.1 ข้อมูลแบบจำนวนเต็ม

ในมุมมองทางคณิตศาสตร์ ข้อมูลแบบจำนวนเต็ม (Integer) ประกอบด้วย จำนวนเต็มบวก จำนวนเต็มศูนย์ และจำนวนเต็มลบ แต่ในมุมมองของการเขียนโปรแกรมภาษาซีนั้น ได้แบ่งจำนวนเต็มออกเป็น 3 ชนิด ประกอบด้วย 1) short integer 2) integer และ 3) long integer แต่ละชนิดมีขนาดและขอบเขตของการใช้งานที่แตกต่างกัน รายละเอียดดังตารางที่ 4.1

ตารางที่ 4.1 ชนิดของข้อมูลแบบจำนวนเต็ม

ชนิดข้อมูล	ขนาด (ไบต์)	เครื่องหมาย	ขอบเขตข้อมูล
short integer	2	signed	-32,768 ถึง 32,768
		unsigned	0 ถึง 65,535
integer	4	signed	-2,147,483,648 ถึง 2,147,483,648

		unsigned	0 ถึง 4,294,967,295
long integer	8	signed	-9,223,372,036,854,775,808 ถึง 9,223,372,036,854,775,807
		unsigned	0 ถึง 18,446,744,073,709,551,615

จากตารางที่ 4.1 พบว่าชนิดข้อมูลแบบ integer และ long integer มีขนาดและขอบเขต ของข้อมูลเหมือนกันทุกประการ เนื่องจากแต่เดิมตามมาตรฐานแอนซีซี (ANSI C) นั้น ชนิดของข้อมูลมีเพียง short integer และ long integer แต่ในการใช้งานจริงกลับพบว่าโปรแกรมเมอร์นิยมใช้แต่ชนิด long integer เพียงอย่างเดียว ดังนั้นจึงมีการเพิ่มชนิดของข้อมูลอีกหนึ่งชนิดในภายหลัง คือข้อมูลชนิด integer โดยมีลักษณะเหมือนกับ long integer ทุกประการ แต่ในการประกาศใช้งานนั้น เขียนสั้นกว่าเพื่ออำนวยความสะดวกให้โปรแกรมเมอร์ในการประกาศตัวแปร จึงเป็นที่มาของ การมีชนิดข้อมูลที่แตกต่างกัน แต่มีขนาดและขอบเขตเหมือนกันดังกล่าว

ตัวอย่างที่ 4.3 การใช้งานข้อมูลแบบจำนวนเต็ม (1)

```
1   #include<stdio.h>
2   void main()
3   {
4       int number1 = 637;
5       int number2 = 308;
6       int plus, minus, multiply;
7
8       plus  = number1 + number2;
9       minus = number1 - number2;
10      multiply = number1 * number2;
11
12      printf("number1 + number2 = %d \n", plus);
13      printf("number1 - number2 = %d \n", minus);
14      printf("number1 x number2 = %d \n", multiply);
15  }
```

ผลลัพธ์ตัวอย่างที่ 4.3

```
number1 + number2 = 945
number1 - number2 = 329
number1 x number2 = 196196
```

คำอธิบายตัวอย่างที่ 4.3

บรรทัดที่ 1 เรียกใช้งานส่วนของเฮดเดอร์ไฟล์ ไลบรารี stdio.h ซึ่งทำหน้าที่จัดการอินพุต และเอาต์พุต

บรรทัดที่ 2 การประกาศฟังก์ชันหลัก

บรรทัดที่ 4	ประกาศตัวแปรชนิดจำนวนเต็มชื่อ number1 และกำหนดค่าเป็น 637
บรรทัดที่ 5	ประกาศตัวแปรชนิดจำนวนเต็มชื่อ number2 และกำหนดค่าเป็น 308
บรรทัดที่ 6	ประกาศตัวแปรชนิดจำนวนเต็มชื่อ plus minus และ multiply
บรรทัดที่ 8	คำนวณค่า number1 + number2 และเก็บไว้ในตัวแปร plus
บรรทัดที่ 9	คำนวณค่า number1 - number2 และเก็บไว้ในตัวแปร minus
บรรทัดที่ 10	คำนวณค่า number1 * number2 และเก็บไว้ในตัวแปร multiply
บรรทัดที่ 11	แสดงผลค่าของตัวแปร plus
บรรทัดที่ 12	แสดงผลค่าของตัวแปร minus
บรรทัดที่ 13	แสดงผลค่าของตัวแปร multiply

<u>หมายเหตุ</u> ตั้งแต่หัวข้อที่ 4.2 เป็นต้นไป จะมีสัญลักษณ์ที่ยังไม่เคยอธิบายวิธีการใช้งานมาก่อน และจะกล่าวถึงโดยละเอียดในบทที่ 6 การแสดงผลและการรับข้อมูล ประกอบด้วย

%d หมายถึง ให้แสดงผลค่าของตัวแปรในรูปแบบตัวเลขจำนวนเต็ม

%c หมายถึง ให้แสดงผลค่าของตัวแปรในรูปแบบตัวอักษร

%f หมายถึง ให้แสดงผลค่าของตัวแปรในรูปแบบตัวเลขจำนวนทศนิยม

%s หมายถึง ให้แสดงผลค่าของตัวแปรในรูปแบบข้อมูลชนิดข้อความ

\n หมายถึง หลังจากแสดงผลแล้วให้เคอเซอร์ขึ้นบรรทัดใหม่

ตัวอย่างที่ 4.4 การใช้งานข้อมูลแบบจำนวนเต็ม (2)

```
#include<stdio.h>
void main()
{
    int width = 38;
    int length  = 69;
    int area;

    area = width * length;

    printf("width = %d \n", width);
    printf("length = %d \n", length);
    printf("area = %d \n", area);
}
```

ผลลัพธ์ตัวอย่างที่ 4.4

```
width = 38
length = 69
area = 2622
```

คำอธิบายตัวอย่างที่ 4.4

บรรทัดที่ 1	เรียกใช้งานส่วนของเฮดเดอร์ไฟล์ ไลบรารี stdio.h ซึ่งทำหน้าที่จัดการอินพุต และเอาต์พุต
บรรทัดที่ 3	การประกาศฟังก์ชันหลัก
บรรทัดที่ 5	ประกาศตัวแปรชนิดจำนวนเต็มชื่อ width เพื่อเก็บค่าความกว้างสี่เหลี่ยม และกำหนดค่าเป็น 38
บรรทัดที่ 6	ประกาศตัวแปรชนิดจำนวนเต็มชื่อ length เพื่อเก็บค่าความยาวสี่เหลี่ยม และกำหนดค่าเป็น 69
บรรทัดที่ 7	ประกาศตัวแปรชนิดจำนวนเต็มชื่อ area เพื่อเก็บค่าพื้นที่สี่เหลี่ยม
บรรทัดที่ 9	คำนวณค่าพื้นทีสี่เหลี่ยมและเก็บไว้ในตัวแปร area
บรรทัดที่ 11 ถึง 13	แสดงผลค่าของตัวแปร 1) width 2) length และ 3) area ตามลำดับ

4.2.2 ข้อมูลแบบจำนวนทศนิยม

ข้อมูลแบบจำนวนทศนิยม (Floating point) สามารถเขียนได้ 2 รูปแบบ เช่น 37.45 และ 2.089E+15 ซึ่งหมายความว่า 2.089×10^{15} ในการเขียนโปรแกรมภาษาซีได้แบ่งจำนวนทศนิยมออกเป็น 3 ชนิด ประกอบด้วย 1) float 2) double และ 3) long double แต่ละชนิดมีขนาดและขอบเขต ของการใช้งานที่แตกต่างกัน โดยมีรายละเอียดดังตารางที่ 4.2

ตารางที่ 4.2 ชนิดของข้อมูลแบบจำนวนทศนิยม

ชนิดข้อมูล	ขนาด (ไบต์)	ขอบเขตข้อมูล
float	4	3.4×10^{-38} ถึง 3.4×10^{38}
double	8	1.7×10^{-308} ถึง 1.7×10^{308}
long double	10	3.4×10^{-4932} ถึง 1.1×10^{4932}

ตัวอย่างที่ 4.5 การใช้งานข้อมูลแบบจำนวนทศนิยม (1)

```
1  #include<stdio.h>
2  #include<math.h>
3  void main()
4  {
5      float radius  = 27.382;
6      float area, circumference;
7  
8      area = M_PI * radius * radius;
9      circumference = 2 * M_PI * radius;
10 
11     printf("radius = %f \n", radius);
12     printf("area = %f \n", area);
13     printf("circumference = %f \n", circumference);
14 }
```

ผลลัพธ์ตัวอย่างที่ 4.5

```
radius = 27.382000
area = 2355.484131
circumference = 172.046173
```

คำอธิบายตัวอย่างที่ 4.5

บรรทัดที่ 1 ถึง 2	เรียกใช้งานส่วนของเฮดเดอร์ไฟล์ ไลบรารี stdio.h ซึ่งทำหน้าที่จัดการอินพุต และเอาต์พุต และไลบรารี math.h ซึ่งทำหน้าที่จัดการทางคณิตศาสตร์ให้ค่าเป็นตัวเลขทศนิยม double
บรรทัดที่ 3	การประกาศฟังก์ชันหลัก
บรรทัดที่ 5	ประกาศตัวแปรชนิดจำนวนทศนิยมชื่อ radius เพื่อเก็บค่ารัศมีของวงกลม และกำหนดค่าเป็น 27.382
บรรทัดที่ 6	ประกาศตัวแปรชนิดจำนวนทศนิยมชื่อ area และ circumference เพื่อเก็บค่าพื้นที่และเส้นรอบวงของวงกลม ตามลำดับ
บรรทัดที่ 8	คำนวณค่าพื้นทีของวงกลมและเก็บไว้ในตัวแปร area โดยค่าของ M_PI ค่าคงที่จากไลบรารี math.h
บรรทัดที่ 9	คำนวณค่าเส้นรอบวงของวงกลมและเก็บไว้ในตัวแปร circumference โดยค่าของ M_PI ค่าคงที่จากไลบรารี math.h
บรรทัดที่ 11 ถึง 13	แสดงผลค่าของตัวแปร 1) radius 2) area และ 3) circumference ตามลำดับ

ตัวอย่างที่ 4.6 การใช้งานข้อมูลแบบจำนวนทศนิยม (2)

```c
1  #include<stdio.h>
2  void main()
3  {
4      double base = 326.9;
5      double height = 500.5;
6      double area;
7
8      area = 0.5 * base * height;
9
10     printf("Base = %f \n", base);
11     printf("Height = %f \n", height);
12     printf("Area of triangle = %f \n", area);
13 }
```

ผลลัพธ์ตัวอย่างที่ 4.6

```
Base = 326.900000
Height = 500.500000
Area of triangle = 81806.725000
```

คำอธิบายตัวอย่างที่ 4.6

บรรทัดที่ 1	เรียกใช้งานส่วนของเฮดเดอร์ไฟล์ ไลบรารี stdio.h ซึ่งทำหน้าที่จัดการอินพุตและเอาต์พุต
บรรทัดที่ 3	การประกาศฟังก์ชันหลัก
บรรทัดที่ 4	ประกาศตัวแปรชนิดจำนวนทศนิยมชื่อ base เพื่อเก็บความยาวฐานของสามเหลี่ยมและกำหนดค่าเป็น 326.9
บรรทัดที่ 5	ประกาศตัวแปรชนิดจำนวนทศนิยมชื่อ height เพื่อเก็บความสูงของสามเหลี่ยมและกำหนดค่าเป็น 500.5
บรรทัดที่ 6	ประกาศตัวแปรชนิดจำนวนทศนิยมชื่อ area เพื่อเก็บค่าพื้นที่สามเหลี่ยม
บรรทัดที่ 8	คำนวณค่าพื้นทีของสามเหลี่ยมและเก็บไว้ในตัวแปร area
บรรทัดที่ 10 ถึง 12	แสดงผลค่าของตัวแปร 1) base 2) height และ 3) area ตามลำดับ

4.2.3 ข้อมูลแบบตัวอักษร

ข้อมูลแบบตัวอักษร (Character) สามารถใช้เก็บตัวอักษรตัวเดียวในรูปแบบ ตัวอักษร ตัวเลข หรือสัญลักษณ์พิเศษ เป็นชนิดข้อมูลที่มีขนาด 1 ไบต์ การกำหนดค่าข้อมูล แบบตัวอักษร จะกำหนดให้อยู่ใน

เครื่องหมาย ‘ ’ (Single quote) เพียง 1 ตัวอักษร ลักษณะสำคัญของข้อมูลแบบตัวอักษร คือ จะไม่สามารถนำข้อมูลไปประมวลผลทางคณิตศาสตร์ได้ เช่น ตัวเลข 8 แตกต่างกับตัวอักษร ‘8’ ดังนั้นจึงไม่สามารถประมวลผลสมการ result = ‘8’ + 1 เป็นต้น

ตัวอย่างที่ 4.7 การใช้งานข้อมูลแบบตัวอักษร

```
1  #include<stdio.h>
2  void main()
3  {
4      char char_test;
5  
6      char_test  = 'A';
7      printf("Character of A = %c \n", char_test);
8      printf("ASCII code of A = %d \n", char_test);
9  
10     char_test = 'AB';
11     printf("Character of AB = %c \n", char_test);
12     printf("ASCII code of AB = %d \n", char_test);
13 }
```

ผลลัพธ์ตัวอย่างที่ 4.7

```
Character of A = A
ASCII code of A = 65
Character of AB = B
ASCII code of AB = 66
```

คำอธิบายตัวอย่างที่ 4.7

บรรทัดที่ 1	เรียกใช้งานส่วนของเฮดเดอร์ไฟล์ ไลบรารี stdio.h ซึ่งทำหน้าที่จัดการอินพุตและเอาต์พุต
บรรทัดที่ 2	การประกาศฟังก์ชันหลัก
บรรทัดที่ 4	ประกาศตัวแปรแบบตัวอักษรชื่อ char_test
บรรทัดที่ 6	กำหนดให้ตัวแปร char_test เก็บค่าตัวอักษร ‘A’
บรรทัดที่ 7 ถึง 8	แสดงผลค่าของตัวแปร char_test และรหัสแอสกีของตัวอักษร ‘A’ ตามลำดับ
บรรทัดที่ 10	กำหนดให้ตัวแปร char_test เก็บค่าตัวอักษร ‘AB’
บรรทัดที่ 11 ถึง 12	แสดงผลค่าของตัวแปร char_test และรหัสแอสกีของตัวอักษร ‘AB’ ตามลำดับ

<u>ข้อสังเกตุ</u> จากบรรทัดที่ 10 ถึง 12 โดยปกติการเก็บค่าของ char จะเก็บเพียง 1 ตัวอัษร หากมีการกำหนดตัวอักษรภายใน ‘ ’ มากกว่า 1 ตัวอักษร จะเก็บอักษรตัวสุดท้ายเท่านั้น

4.2.4 ข้อมูลแบบข้อความ

ในความเป็นจริงการเขียนโปรแกรมภาษาซีไม่มีประเภทของข้อมูลที่สนับสนุนการเก็บข้อความ แต่สามารถประยุกต์หลักการของข้อมูลแบบอาร์เรย์ (Array) (กล่าวถึงรายละเอียดในบทที่ 10) การกำหนดค่าข้อมูลแบบข้อความจะกำหนดให้อยู่ในเครื่องหมาย “ ” (Double quotes) ลักษณะสำคัญของข้อมูลแบบข้อความคือ ถ้าไม่มีการกำหนดค่าให้กับตัวแปรตั้งแต่เริ่มแรก ต้องประกาศขนาดของอาร์เรย์ ไว้ล่วงหน้าเพื่อทำการจองพื้นที่ของหน่วยความจำในการเก็บข้อมูลของตัวแปร

ตัวอย่างที่ 4.8 การใช้งานข้อมูลแบบข้อความ

```
1  #include<stdio.h>
2  void main()
3  {
4      char string_test[13] = "C Programming";
5
6      printf("String = %s \n", string_test);
7      printf("Index number 3 of string = %c \n", string_test[3]);
8  }
```

ผลลัพธ์ตัวอย่างที่ 4.8

```
String = C Programming
Index number 3 of string = r
```

คำอธิบายตัวอย่างที่ 4.8

บรรทัดที่ 1	เรียกใช้งานส่วนของเฮดเดอร์ไฟล์ ไลบรารี stdio.h ซึ่งทำหน้าที่จัดการอินพุตและเอาต์พุต
บรรทัดที่ 2	การประกาศฟังก์ชันหลัก
บรรทัดที่ 4	ประกาศตัวแปรแบบข้อความชื่อ string_test และกำหนดให้เก็บข้อความ “C Programming”
บรรทัดที่ 6 ถึง 7	แสดงผลค่าของตัวแปรข้อความโดย %s และตัวอักษรลำดับที่ 3 โดย %c ของตัวแปร string_test ตามลำดับ

4.2.5 ข้อมูลแบบค่าคงที่

ข้อมูลแบบค่าคงที่ (Constant) คือข้อมูลที่ไม่มีการเปลี่ยนค่าในขณะที่โปรแกรมกำลังทำงาน เช่น ค่า π ซึ่งมีค่าเท่ากับ 22/7 หรือประมาณ 3.14 เป็นต้น การประกาศค่าคงที่เพื่อใช้งานในการเขียนโปรแกรมภาษาซีสามารถทำได้ 2 รูปแบบ คือ การกำหนดค่าโดยใช้คำสั่ง #define และการกำหนดค่าไว้ในตัวแปร

การกำหนดค่าโดยใช้คำสั่ง #define เป็นการประกาศค่าคงที่ไว้ในส่วนของปรีโปรเซสเซอร์ เช่น ต้องการประกาศค่าคงที่ของภาษีมูลค่าเพิ่มสามารถทำได้โดยใช้คำสั่ง #define vat 0.07 เป็นต้น (ดูตัวอย่างที่ 4.9 เพิ่มเติม)

การกำหนดค่าไว้ในตัวแปร เป็นการประกาศค่าคงที่ให้กับตัวแปร มีลักษณะเหมือนกับการประกาศตัวแปรทั่วไป แต่ต้องขึ้นต้นด้วยการใช้คำสั่ง const เช่น const float vat = 0.07 เป็นต้น (ดูตัวอย่างที่ 4.10 เพิ่มเติม)

ตัวอย่างที่ 4.9 การกำหนดค่าคงที่โดยใช้คำสั่ง #define

```
1   #include<stdio.h>
2
3   #define PI 3.14159
4
5   void main()
6   {
7       float radius = 12.7;
8       float area;
9
10      area = PI * radius * radius;
11
12      printf("Radius = %f \n", radius);
13      printf("Area of circle = %f \n", area);
14  }
```

ผลลัพธ์ตัวอย่างที่ 4.9

```
Radius = 12.700000
Area of circle = 506.707031
```

คำอธิบายตัวอย่างที่ 4.9

บรรทัดที่ 1	เรียกใช้งานส่วนของเฮดเดอร์ไฟล์ ไลบรารี stdio.h ซึ่งทำหน้าที่จัดการอินพุตและเอาต์พุต
บรรทัดที่ 3	ประกาศค่าคงที่โดยใช้คำสั่ง #define กำหนดให้ PI มีค่า 3.14159
บรรทัดที่ 5	การประกาศฟังก์ชันหลัก
บรรทัดที่ 7	ประกาศตัวแปรแบบจำนวนทศนิยมชื่อ radius และกำหนดค่าเป็น 12.7
บรรทัดที่ 8	ประกาศตัวแปรแบบจำนวนทศนิยมชื่อ area
บรรทัดที่ 10	คำนวณค่าพื้นที่ของวงกลมและเก็บไว้ในตัวแปร area
บรรทัดที่ 12 ถึง 13	แสดงผลค่าของตัวแปร 1) radius และ 2) area ตามลำดับ

ตัวอย่างที่ 4.10 การกำหนดค่าคงที่ไว้ในตัวแปร

```
1  #include<stdio.h>
2
3  void main()
4  {
5      const float PI = 3.14;
6      float radius = 12.7;
7      float area;
8
9      area = PI * radius * radius;
10
11     printf("Radius = %f \n", radius);
12     printf("Area of circle = %f \n", area);
13
14 }
```

ผลลัพธ์ตัวอย่างที่ 4.10

```
Radius = 12.700000
Area of circle = 506.450592
```

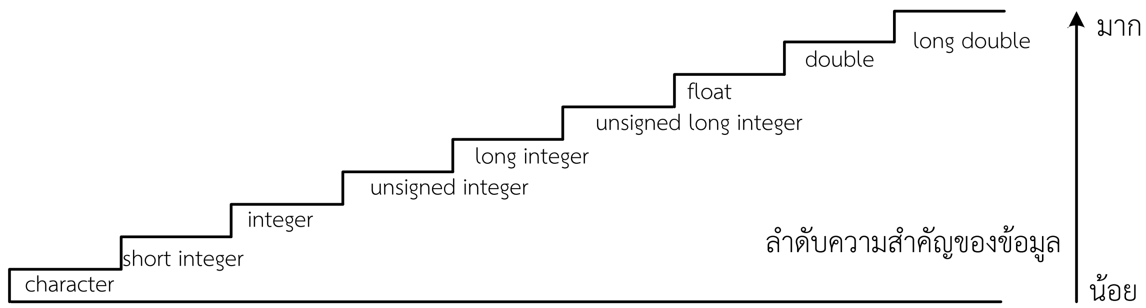
คำอธิบายตัวอย่างที่ 4.10

บรรทัดที่ 1 ถึง 2	เรียกใช้งานส่วนของเฮดเดอร์ไฟล์ ไลบรารี stdio.h ซึ่งทำหน้าที่จัดการอินพุตและเอาต์พุต
บรรทัดที่ 4	การประกาศฟังก์ชันหลัก
บรรทัดที่ 7	ประกาศค่าคงที่แบบจำนวนทศนิยมชื่อ PI โดยใช้คำสั่ง const และกำหนดค่าเป็น 3.14
บรรทัดที่ 8	ประกาศตัวแปรแบบจำนวนทศนิยมชื่อ radius และกำหนดค่าเป็น 12.7
บรรทัดที่ 9	ประกาศตัวแปรแบบจำนวนทศนิยมชื่อ area
บรรทัดที่ 11	คำนวณค่าพื้นที่ของวงกลมและเก็บไว้ในตัวแปร area
บรรทัดที่ 13 ถึง 14	แสดงผลค่าของตัวแปร 1) radius และ 2) area ตามลำดับ

4.3 การแปลงชนิดของข้อมูล

ในการเขียนโปรแกรมภาษาซีเพื่อการคำนวณทางคณิตศาสตร์ บ่อยครั้งข้อมูลที่นำมาคำนวณเป็นคนละประเภทกัน ดังนั้นจึงต้องมีการแปลงชนิดของข้อมูลให้เป็นประเภทเดียวกันเพื่อป้องกัน ความผิดพลาดของผลลัพธ์ในการคำนวณ ภาษาซีมีวิธีการสนับสนุนการแปลงชนิดของข้อมูล 2 แบบประกอบด้วย การแปลงชนิดโดยปริยาย (Implicit type conversion) คือการแปลงชนิดข้อมูล โดยอัตโนมัติตามลำดับ

ความสำคัญของชนิดข้อมูล (ดูรูปที่ 4.5) และ การแปลงชนิดโดยตรง (Explicit type conversion) คือการแปลงชนิดข้อมูลตามความต้องการของโปรแกรมเมอร์ ซึ่งมีรายละเอียดดังนี้



รูปที่ 4.5 ลำดับความสำคัญของข้อมูลในภาษาซี

4.3.1 การแปลงชนิดโดยปริยาย

หลักการทำงานของการแปลงชนิดโดยปริยาย (Implicit type conversion) คือ คอมไพเลอร์จะทำการแปลงประเภทข้อมูลที่มีลำดับความสำคัญต่ำ ไปเป็นข้อมูลที่มีลำดับความสำคัญสูงกว่าโดยอัตโนมัติ ภาษาซีแบ่งลำดับความสำคัญของข้อมูลเป็นหลายระดับ ข้อมูลที่มีลำดับความสำคัญต่ำที่สุดคือข้อมูลแบบตัวอักษร และข้อมูลที่มีลำดับความสำคัญสูงที่สุดคือข้อมูลแบบจำนวนทศนิยมชนิด long double

ตัวอย่างที่ 4.11 การทำงานของการแปลงชนิดโดยปริยาย (1)

นิพจน์	คำอธิบาย
integer + long integer	แปลง integer เป็น long integer
char – float	แปลง char เป็น float
float - long double	แปลง float เป็น long double
integer / double	แปลง integer เป็น double
(integer + float) * double	ทำในวงเล็บก่อนโดยแปลง integer เป็น float จากนั้นแปลง float เป็น double

ตัวอย่างที่ 4.12 การทำงานของการแปลงชนิดโดยปริยาย (2)

```
1  #include<stdio.h>
2  
3  void main()
4  {
5      int int_test = 10;
6      char char_test = 'A';
7  
8      printf("Before data type conversion \n");
9      printf("int_test contain : %d \n ", int_test);
10     printf("char_test contain : %c \n\n ", char_test);
11 
12     int_test = int_test + char_test;
13 
14     printf("After data type conversion \n");
15     printf("int_test contain : %d \n ", int_test);
16     printf("char_test contain : %c \n ", char_test);
17 }
```

ผลลัพธ์ตัวอย่างที่ 4.12

```
Before data type conversion
int  test contain : 10
char  test contain : A

After data type conversion
int  test contain : 75
char_test contain : A
```

คำอธิบายตัวอย่างที่ 4.12

บรรทัดที่ 1	เรียกใช้งานส่วนของเฮดเดอร์ไฟล์ ไลบรารี stdio.h ซึ่งทำหน้าที่จัดการอินพุตและเอาต์พุต
บรรทัดที่ 3	การประกาศฟังก์ชันหลัก
บรรทัดที่ 5	ประกาศตัวแปรแบบจำนวนเต็มชื่อ int_test และกำหนดค่าเป็น 10
บรรทัดที่ 6	ประกาศตัวแปรแบบตัวอักษรชื่อ char_test และกำหนดค่าเป็น ’A’
บรรทัดที่ 8	แสดงผลข้อความ “Before data type conversion”
บรรทัดที่ 9 ถึง 10	แสดงผลค่าของตัวแปร 1) int_test และ 2) char_test ตามลำดับ
บรรทัดที่ 12	นำค่าในตัวแปร int_test บวกกับค่าในตัวแปร char_test และเก็บไว้ในตัวแปร int_test
บรรทัดที่ 14	แสดงผลข้อความ “After data type conversion”
บรรทัดที่ 15 ถึง 16	แสดงผลค่าของตัวแปร 1) int_test และ 2) char_test ตามลำดับ

4.3.2 การแปลงชนิดโดยตรง

หลักการทำงานของการแปลงชนิดโดยตรง (Explicit type conversion) คือ โปรแกรมเมอร์จะกำหนดชนิดของข้อมูลปลายทางที่ต้องการ โดยไม่ขึ้นกับลำดับความสำคัญของข้อมูล ศึกษาการทำงานของการแปลงข้อมูลแบบ Explicit type conversion จากตัวอย่างที่ 4.12 และ 4.13

ตัวอย่างที่ 4.13 การทำงานของการแปลงชนิดโดยตรง (1)

กำหนดให้ ตัวแปร A มีชนิดข้อมูลเป็น integer

ตัวแปร B มีชนิดข้อมูลเป็น long integer

ตัวแปร C มีชนิดข้อมูลเป็น float

ตัวอย่าง	คำอธิบาย
C = (float) A	แปลงชนิดข้อมูลตัวแปร A จาก integer เป็น float และนำผลลัพธ์ที่ได้เก็บในตัวแปร C
A = (int) B	แปลงชนิดข้อมูลตัวแปร B จาก long integer เป็น integer แสนำผลลัพธ์เก็บในตัวแปร A
C = (float) (A + B)	นำตัวแปร A + B จากนั้นแปลงผลลัพธ์ที่ได้เป็น float และนำผลลัพธ์เก็บในตัวแปร C
C = (float) A / B	แปลงชนิดข้อมูลตัวแปร A จาก integer เป็น float โดยวิธีการ explicit type conversion จากนั้นแปลงชนิดข้อมูลตัวแปร B จาก long integer เป็น float โดยวิธีการ implicit type conversion และนำผลลัพธ์เก็บในตัวแปร C

ตัวอย่างที่ 4.14 การทำงานของการแปลงชนิดโดยตรง (2)

```
#include<stdio.h>
void main()
{
    int int_test = 65;
    double double_test = 66;
    char char_test ;

    char_test = (char)int_test;
    printf("int_test = %d \n", int_test);
    printf("Assign char_test = (char)int_test \n");
    printf("char_test = %c \n\n", char_test);

    char_test = (char)double_test;
    printf("double_test = %f \n", double_test);
    printf("Assign char_test = (char)double_test \n");
    printf("char_test = %c \n", char_test);
}
```

ผลลัพธ์ตัวอย่างที่ 4.14

```
int_test = 65
Assign char_test = (char)int_test
char_test = A

double_test = 66.000000
Assign char_test = (char)double_test
char_test = B
```

คำอธิบายตัวอย่างที่ 4.14

บรรทัดที่ 1	เรียกใช้งานส่วนของเฮดเดอร์ไฟล์ ไลบรารี stdio.h ซึ่งทำหน้าที่จัดการอินพุตและเอาต์พุต
บรรทัดที่ 2	การประกาศฟังก์ชันหลัก
บรรทัดที่ 4	ประกาศตัวแปรแบบจำนวนเต็มชื่อ int_test และกำหนดค่าเป็น 65
บรรทัดที่ 5	ประกาศตัวแปรแบบจำนวนทศนิยมชื่อ double_test และกำหนดค่าเป็น 66
บรรทัดที่ 6	ประกาศตัวแปรแบบตัวอักษรชื่อ char_test
บรรทัดที่ 8	แปลงชนิดข้อมูลตัวแปร int_test จาก integer เป็น character เก็บผลลัพธ์ที่ได้ในตัวแปร char_test
บรรทัดที่ 9	แสดงผลค่าของตัวแปร int_test
บรรทัดที่ 10	แสดงผลข้อความ “Assign char_test = (char)int_test”
บรรทัดที่ 11	แสดงผลค่าของตัวแปร char_test
บรรทัดที่ 13	แปลงชนิดข้อมูลตัวแปร double_test จาก double เป็น character เก็บผลลัพธ์ที่ได้ในตัวแปร char_test
บรรทัดที่ 14	แสดงผลค่าของตัวแปร double_test
บรรทัดที่ 15	แสดงผลข้อความ “Assign char_test = (char)double_test”
บรรทัดที่ 16	แสดงผลค่าของตัวแปร char_test

สรุปเนื้อหาบทที่ 4

ตัวแปรคือสัญลักษณ์ที่กำหนดขึ้นเพื่อใช้แทนค่าของข้อมูลในการเขียนโปรแกรม ซึ่งข้อมูลที่เก็บในตัวแปรนั้นจะเป็นตัวหนังสือหรือตัวเลขก็ได้ การตั้งชื่อตัวแปรที่ดีนั้นมีความสำคัญในการเขียนโปรแกรม ชื่อตัวแปรที่ดีช่วยสื่อถึงหน้าที่ของตัวแปร ชื่อตัวแปรที่ดีจะต้องไม่สั้นและยาวจนเกินไป ชื่อตัวแปรที่สั้นจะไม่สื่อความหมายถึงหน้าที่ของตัวแปร ส่วนชื่อตัวแปรที่ยาวถึงแม้ จะสื่อถึงหน้าที่ของตัวแปรได้ดีกว่าแต่อาจจะทำให้โปรแกรมอ่านยากกว่าชื่อตัวแปรที่สั้นกว่า

ตัวแปรในภาษาซีสามารถแบ่งได้ 2 ประเภท ประกอบด้วย **ตัวแปรแบบโลคอล** คือตัวแปรที่ประกาศไว้ภายในฟังก์ชันใดฟังก์ชันหนึ่ง ส่งผลให้ฟังก์ชันนั้นเป็นเจ้าของตัวแปรที่ประกาศ โดยที่ฟังก์ชันอื่นไม่สามารถเรียกใช้งานตัวแปรนั้นได้ **ตัวแปรแบบโกลบอล** คือตัวแปรที่ประกาศ ไว้ภายนอกฟังก์ชัน ส่งผลให้สามารถเรียกใช้งานตัวแปรนั้นได้จากทุกส่วนในโปรแกรมต้นฉบับ

การเขียนโปรแกรมภาษาซี มีประเภทของข้อมูลให้เลือกใช้อย่างหลากหลาย ข้อมูลแต่ละประเภทมีข้อดี ข้อเสีย และความเหมาะสมในการนำไปใช้งานแตกต่างกัน ข้อมูลที่มีความกว้าง ของขอบเขตมากก็มีความต้องการพื้นที่ของหน่วยความจำมากเช่นกัน ข้อมูลแบ่งเป็น 5 ประเภท ประกอบด้วย **ข้อมูลแบบจำนวนเต็ม** ประกอบด้วยจำนวนเต็มบวก จำนวนเต็มศูนย์ และจำนวนเต็มลบ **ข้อมูลแบบจำนวนทศนิยม** สามารถเขียนได้ 2 รูปแบบ เช่น 37.45 และ 2.089E+15 ซึ่งหมายความว่า 2.089×10^{15} **ข้อมูลแบบตัวอักษร** สามารถใช้เก็บตัวอักษรตัวเดียวในรูปแบบ ตัวอักษร ตัวเลข หรือสัญลักษณ์พิเศษ **ข้อมูลแบบข้อความ** การเขียนโปรแกรมภาษาซีไม่มีประเภทของข้อมูลที่สนับสนุนการเก็บข้อความ แต่สามารถประยุกต์หลักการของข้อมูลแบบอาร์เรย์ เพื่อสร้างข้อมูลแบบข้อความได้ และ **ข้อมูลแบบค่าคงที่** คือข้อมูลที่ไม่มีการเปลี่ยนค่าในขณะที่โปรแกรมกำลังทำงาน

ภาษาซีมีวิธีการสนับสนุนการแปลงชนิดของข้อมูล 2 แบบประกอบด้วย **การแปลงชนิดโดยปริยาย** (Implicit type conversion) คือการแปลงชนิดข้อมูลโดยอัติโนมัติตามลำดับความสำคัญของชนิดข้อมูล และ **การแปลงชนิดโดยตรง** (Explicit type conversion) คือการแปลงชนิดข้อมูล ตามความต้องการของโปรแกรมเมอร์

บทที่ 5

การดำเนินงานทางคณิตศาสตร์และตรรกศาสตร์

5.1 ตัวดำเนินการและนิพจน์ (Operators and Expressions)

ในการเขียนโปรแกรมองค์ประกอบที่สำคัญสิ่งหนึ่งคือ ตัวดำเนินการ (Operator) ซึ่งมักจะนำมาใช้ร่วมกับนิพจน์ ในลักษณะต่างๆ ได้ดังนี้

5.1.1 ตัวดำเนินการคณิตศาสตร์ (Mathematical Operators) เป็นตัวดำเนินการทางด้านคณิตศาสตร์ถือว่าเป็นดำเนินการพื้นฐานที่สุดทั้งในชีวิตประจำวันและในการเขียนโปรแกรม ซึ่งถือได้ว่าเป็นการดำเนินการที่ต้องกระทำอยู่บ่อยครั้ง โดยเครื่องหมายที่ใช้ในการคำนวณทางคณิตศาสตร์ในภาษา C จะแสดงดังตารางที่ 5.1

ตาราง 5.1 ตัวดำเนินการคณิตศาสตร์

เครื่องหมาย	ตัวดำเนินการ	การใช้งาน	ความหมาย
+	บวก	z = x + y;	บวกค่าในตัวแปร x กับค่าในตัวแปร y ผลลัพธ์เก็บไว้ที่ตัวแปร z
-	ลบ	z = x - y;	ลบค่าในตัวแปร x ด้วยค่าในตัวแปร y ผลลัพธ์เก็บไว้ที่ตัวแปร z
*	คูณ	z = x * y;	คูณค่าในตัวแปร x ด้วยค่าในตัวแปร y ผลลัพธ์เก็บไว้ที่ตัวแปร z
/	หาร	z = x / y;	หารค่าในตัวแปร x ด้วยค่าในตัวแปร y ผลลัพธ์เก็บไว้ที่ตัวแปร z
%	หารเอาเศษ (Modulo)	z = x % y;	หารค่าในตัวแปร x ด้วยค่าในตัวแปร y ผลลัพธ์คือ เศษที่ได้จากการหาร เก็บไว้ที่ตัวแปร z

ตัวอย่างที่ 1 การใช้เครื่องหมายการคำนวณทางคณิตศาสตร์ในภาษา C

```
#include<stdio.h>
#include<conio.h>
int x = 9, y = 6;              //สร้างตัวแปร x กำหนดค่าเริ่มต้นเป็น 5 และ y เป็น 2
int sum,diff,mult,mod;         //สร้างตัวแปรชนิด int เพื่อเก็บตัวเลขจำนวนเต็ม
float div;                     //สร้างตัวแปรชนิด float เพื่อเก็บตัวเลขจำนวนเต็ม
void main(){
    sum = x+y;                 //สร้างตัวแปรชนิด float เพื่อเก็บตัวเลขจำนวนเต็ม
    diff = x-y;                //ลบค่าของตัวแปร x ด้วยค่าของ y ผลลัพธ์เก็บไว้ที่ diff
    mult = x*y;                //คูณค่าของตัวแปร x กับ y ผลลัพธ์เก็บไว้ที่ mult
    div = x/y;                 //หารค่าของตัวแปร x ด้วยค่าของ y ผลลัพธ์เก็บไว้ที่ div
    mod = x%y;                 //ตัวแปร mod เก็บผลลัพธ์เศษจากการหารค่าของ x ด้วย y
    printf("sum = %d\n",sum);
    printf("diff = %d\n",diff);
    printf("mult = %d\n",mult);    //แสดงผลลัพธ์จากการคำนวณออกทางจอภาพ
    printf("div = %f\n",div);
    printf("mod = %d\n",mod);
}
```

เมื่อสั่งโปรแกรมทำงานผลโปรแกรมตัวอย่างที่ 1 จะปรากฏผลการทำงานดังต่อไปนี้

```
sum = 15
diff = 3
mult = 54
div = 1.000000
```

1) เครื่องหมายการคำนวณทางคณิตศาสตร์ประเภทเพิ่มหรือลดค่า

เครื่องหมายการคำนวณทางคณิตศาสตร์ประเภทของการเพิ่มหรือลดค่าลงทีละหนึ่ง แสดงวิธีการดังตารางที่ 5.2

ตาราง 5.2 เครื่องหมายการคำนวณทางคณิตศาสตร์ประเภทเพิ่มหรือลดค่า

เครื่องหมาย	ตัวดำเนินการ	การใช้งาน	ความหมาย
++	เพิ่มค่าทีละหนึ่ง (Increment)	z = ++x;	บวกค่าในตัวแปร x เพิ่มขึ้น 1 ก่อนที่จะกำหนดค่าให้กับตัวแปร z
		z = x++;	กำหนดค่าของตัวแปร x ให้กับตัวแปร z ก่อนที่จะเพิ่มค่าของตัวแปร x ขึ้นอีก 1

--	ลดค่าทีละหนึ่ง (Decrement)	z = --x;	ลดค่าในตัวแปร x ลง 1 ก่อนที่จะกำหนดค่าให้กับตัวแปร z
		z = x--;	กำหนดค่าของตัวแปร x ให้กับตัวแปร z ก่อนที่จะลดค่าของตัวแปร x ลง 1

สำหรับเครื่องหมายประเภทเพิ่มหรือลดค่านี้ต้องระวังลำดับก่อนหลังของเครื่องหมายให้ดีถ้าเครื่องหมายอยู่ด้านหน้าตัวแปร เช่น z =++x; หรือ z =--x; นั่นหมายถึง ให้เพิ่มหรือลด ค่าของ x ก่อนที่จะกำหนดค่าให้กับตัวแปร z ส่วนถ้าเครื่องหมายอยู่ด้านหลัง เช่น z = x++; หรือ z =x--; คือ การกำหนดค่าของ x ให้กับ z ก่อนที่จะเพิ่มหรือลดค่าของ x เพื่อความเข้าใจให้ดูผลลัพธ์ของโปรแกรมต่อไปนี้

ตัวอย่างที่ 2 การใช้เครื่องหมายประเภทเพิ่มหรือลดค่า

```
#include<stdio.h>
#include<conio.h>
int x = 4, y = 4, z = 0;          //กำหนดค่าตัวแปร x และ y ให้มีค่าเท่ากับ 4 และ 0
void main(){
    z = ++x;                      //บวกค่าของตัวแปร x เพิ่มขึ้น 1 ก่อนที่จะกำหนดให้กับ z
    printf("z = %d\n",z);         //แสดงผลตัวแปร z ซึ่งจะต้องมีค่าเท่ากับ 5
    printf("x = %d\n",x);         //แสดงผลตัวแปร x ซึ่งจะต้องมีค่า 5 เท่ากับ z
    z = y++;                      //กำหนดค่าของ y ให้กับ z ก่อนที่จะบวกค่าของ y เพิ่มขึ้น 1
    printf("z = %d\n",z);         //แสดงผลตัวแปร z ซึ่งจะต้องมีค่าเท่ากับ y ก่อนเพิ่มค่าซึ่งก็คือ 4
    printf("y = %d\n",y);         //แสดงผลตัวแปร y ซึ่งจะต้องมีค่าเท่ากับ
}
```

เมื่อสั่งโปรแกรมทำงานผลโปรแกรมตัวอย่างที่ 2 จะปรากฏผลการทำงานดังต่อไปนี้

```
z = 5
x = 5
z = 4
y = 5
```

2) เครื่องหมายการคำนวณทางคณิตศาสตร์ประเภทลดรูป

นอกจากเครื่องหมายคำนวณตามปกติ และเครื่องหมายคำนวณแบบเพิ่ม/ลดค่าแล้ว การคำนวณทางคณิตศาสตร์ในภาษา C ยังมีเครื่องหมายประเภทที่เรียกว่า ลดรูปอีกด้วย ดังแสดงรายละเอียดในตารางที่ 5.3

ตาราง 5.3 เครื่องหมายการคำนวณทางคณิตศาสตร์ประเภทลดรูป

เครื่องหมาย	ตัวดำเนินการ	การใช้งาน	ความหมาย
+=	บวกแบบลดรูป	z += x;	บวกค่าในตัวแปร z กับค่าของตัวแปร x ผลลัพธ์จะนำไปเก็บไว้ที่ตัวแปร z มีค่าเหมือนกับ z=z+x
-=	ลบแบบลดรูป	z -= x;	ลบค่าในตัวแปร z ด้วยค่าของตัวแปร x ผลลัพธ์จะนำไปเก็บไว้ที่ตัวแปร z มีค่าเหมือนกับ z=z-x
*=	คูณแบบลดรูป	z *= x;	คูณค่าในตัวแปร z กับค่าของตัวแปร x ผลลัพธ์จะนำไปเก็บไว้ที่ตัวแปร z มีค่าเหมือนกับ z=z*x
/=	หารแบบลดรูป	z /= x;	หารค่าในตัวแปร z ด้วยค่าของตัวแปร x ผลลัพธ์จะนำไปเก็บไว้ที่ตัวแปร z มีค่าเหมือนกับ z=z/x
%=	หารเอาเศษ แบบลดรูป	z %= x;	หารค่าในตัวแปร z ด้วยค่าของตัวแปร x ผลลัพธ์คือเศษที่ได้จากการหารนำไปเก็บไว้ที่ตัวแปร z มีค่าเหมือนกับ z=z%x

5.1.2 ตัวดำเนินการเปรียบเทียบ (Relational Operators) หมายถึงเครื่องหมายที่ใช้ในการเปรียบเทียบและตัดสินใจ ผลการเปรียบเทียบจะให้ค่าเป็น 1 และเท็จจะให้ค่าเป็น 0 เครื่องหมายที่ใช้แสดงดังตารางที่ 5.4

ตาราง 5.4 ตัวดำเนินการเปรียบเทียบ

เครื่องหมาย	ตัวดำเนินการ	ตัวอย่าง
>	มากกว่า	A > B (A มากกว่า B)
>=	มากกว่าหรือเท่ากับ	A >= B (A มากกว่าหรือเท่ากับ B)
<	น้อยกว่า	A < B (A กว่า B)
<=	น้อยกว่าหรือเท่ากับ	A <= B (A กว่าหรือเท่ากับ B)

เครื่องหมาย	ตัวดำเนินการ	ตัวอย่าง
==	เท่ากับ	A == B (A กับ B)
!=	ไม่เท่ากับ	A != B (A ไม่กับ B)

5.1.3 ตัวดำเนินการตรรกะ (logical operators) หมายถึงเครื่องหมายตรรกศาสตร์ที่ใช้ในการหาผลลัพธ์โดยใช้หลักการทางตรรกศาสตร์ส่วนใหญ่มักใช้ในระบบคอมพิวเตอร์หรือระบบเกี่ยวกับวงจรไฟฟ้า โดยเครื่องหมายทางตรรกศาสตร์ที่ใช้ในการเขียนโปรแกรมภาษา c มีอยู่ 3 ชนิดคือ && (and), || (or) และ ! (not) ที่ใช้งานแสดงดังตารางที่ 5.5

ตาราง 5.5 ตัวดำเนินการตรรกะ

X	Y	X&&Y (AND)	X||Y (OR)	!X (NOT)
F (0)	F (0)	F (0)	F (0)	T (1)
F (0)	T (1)	F (0)	T (1)	T (1)
T (1)	F (0)	F (0)	T (1)	F (0)
T (1)	T (1)	T (1)	T (1)	F (0)

ตาราง 5.6 ตัวอย่างการเปรียบเทียบตัวดำเนินการตรรกะ เมื่อ สมมติให้ int a=1, b=3, c=4

นิพจน์	แปลงนิพจน์	ค่าตรรกะ	ค่าที่ได้
a + b > 0 && b / c > 1	=((a+b)>0) && ((b/c) > 1) =((1 + 3) > 0) && ((3 / 4) > 1) =(4 > 0) && (0.75 > 1) =T && F	False	0
a /c == b / c && (a+b) /c ==1	=((a/c)==(b/c))&&(((a+b)/c)==1) =((1/4)==(3/4))&&(((1+3)/4)==1) =(0.25==0.75)&&((4/4)==1) =F && T	False	0

นิพจน์	แปลงนิพจน์	ค่าตรรกะ	ค่าที่ได้
a != b/2 || c<=b && ! (c==a%2)	=(a != (b/2)) || (c <= b) && (!(c==a%2)) =(1 != (3/2)) || (4 <= 3) && (!(4==1%2)) =(1 != 1.5) || (4 <= 3) && (!(4==1)) =T || F && T =T || F	True	1

ขั้นตอนการทำงานของตัวดำเนินการในภาษา C

ในบางครั้งนิพจน์ประกอบด้วยตัวดำเนินการมากมาย ทำให้เกิดความยุ่งยากในการพิจารณา ขั้นตอนการทำงานของตัวดำเนินการ จึงได้ตั้งกฎเกี่ยวกับลำดับการทำงานก่อนหลังของตัวดำเนินการ (Operator) ดังตารางที่ 5.7

ตาราง 5.7 ลำดับการทำงานก่อนหลังของตัวดำเนินการ

ตัวดำเนินการ	ลำดับความสำคัญในการทำงาน
( )	1 (สูงสุด)
! , ++ , --	2
* , / , %	3
+ , -	4
< , <= , > , >=	5
== , !=	6
&&	7
||	8
+=, -= , *= , = , /= , %=	9 (ต่ำสุด)

ตัวดำเนินการที่มีความสำคัญอยู่ในระดับเดียวกัน ให้ทำงานตามขั้นตอนจากซ้ายไปขวาเป็นหลักต่อไปนี้ดังตัวอย่างต่อไปนี้

- 5 + 7 * 2 = 5 + 14

  = 19

จะสังเกตุเห็นว่า 5 + 7 * 2 มีตัวดำเนินการ * อยู่ลำดับสูงกว่า + จึงต้องคูณเลขก่อนแล้วจึงบวกเลขในภายหลัง

- 2 * 3 – 14 / 7 + 5 = 6 – 14 / 7 + 5

  = 6 – 2 + 5

  = 4 + 5

  = 9

จะสังเกตุเห็นว่า 2 * 3 – 14 / 7 + 5 มี ตัวดำเนินการ * และ / อยู่ลำดับเดียวกัน และมีลำดับความสำคัญมากกว่า ตัวดำเนินการ – และ + จึงทำให้ ดำเนินการคูณเลขแล้วหารเลขก่อน เพราะตัวดำเนินการ * และ / มีความสำคัญอยู่ในระดับเดียวกันให้ทำจากซ้าย ไปขวา จากนั้น ตัวดำเนินการ – และ + ก็อยู่ในลำดับเดียวกันจึงทำจากซ้ายมือ ก่อนคือ ลบเลข แล้วจึงบวกเลขในภายหลัง

5.1.4 ตัวดำเนินการ sizeof (Sizeof Operator) หมายถึง ตัวดำเนินการที่ใช้เพื่อหาขนาดของตัวแปรในภาษา C โดยตัวแปรเหล่านี้สามารถเป็นตัวแปรประเภทใดก็ได้ นอกจากนี้ยังสามารถใช้เพื่อค้นหาขนาดของประเภทข้อมูลได้อีกด้วย ซึ่งมีได้ 3 รูปแบบดังนี้

sizeof (data-type);	// ชนิดข้อมูลพื้นฐาน เช่น int, float, double
sizeof (object);	// ชนิดที่สร้างใหม่ เช่น class
sizeof object;	// ชนิดที่สร้างใหม่ แบบไม่ต้องใส่วงเล็บ

ตัวอย่างที่ 3 การใช้ตัวดำเนินการ sizeof

```c
#include<stdio.h>
void main(){
    printf("char = %d \n",sizeof(char));
    printf("unsigned char = %d \n", sizeof(unsigned char));
    printf("int = %d \n", sizeof(int));
    printf("unsigned int = %d \n", sizeof(unsigned int));
    printf("float = %d \n", sizeof(float));
    printf("double = %d \n", sizeof(double));
    int x=5;
    printf("Size of x = %d \n",sizeof(x));
}
```

เมื่อสั่งโปรแกรมทำงานผลโปรแกรมตัวอย่างที่ 3 จะปรากฏผลการทำงานดังต่อไปนี้

```
char = 1
unsigned char = 1
int = 4
unsigned int = 4
float = 4
double = 8
Size of x = 4
```

5.1.5 ตัวดำเนินการจุลภาค (Comma Operator) หมายถึง ตัวดำเนินการที่ใช้คั่นระหว่างนิพจน์ต่างๆ ค่านิพจน์ต่างๆ นั้นจะถูกตีความหมายจากนิพจน์ซ้ายสุดไปยังนิพจน์ขวาสุด ผลลัพธ์ของการใช้ตัวดำเนินการ , (Comma) จะเป็นนิพจน์ขวาสุด

นอกจากนี้เราใช้ตัวดำเนินการ , ในที่อื่น ๆ ได้อีก เช่น การใช้คั่นระหว่างพารามิเตอร์หรืออาร์กิวเมนต์ของฟังก์ชันซึ่งจะใช้ร่วมกับตัวดำเนินการวงเล็บด้วย

5.1.6 ตัวดำเนินการเทอร์นารี่ (Ternary Operator) หมายถึง ตัวดำเนินการนี้จะเรียกว่าเทอร์นารี่เพราะว่าใช้ถึงสามนิพจน์ในการประมวลผล (Ternary มีความหมายเท่ากับ 3) ซึ่งจะเหมือนกับการใช้คำสั่งเงื่อนไข if – else รูปแบบการใช้โอเปอร์เรเตอร์เทอร์นารี่ มีรูปแบบดังนี้

นิพจน์แรก ? นิพจน์ที่สอง : นิพจน์ที่สาม;

ผลลัพธ์ของการใช้ตัวดำเนินการนี้ จะเป็นนิพจน์ที่ 2 หรือ 3 ซึ่งจะขึ้นอยู่กับนิพจน์แรก ถ้านิพจน์แรกเป็นจริง ผลลัพธ์จะเป็นนิพจน์ที่สอง ถ้านิพจน์ แรกเป็นเท็จ ผลลัพธ์จะเป็นนิพจน์ที่สามซึ่งจะเทียบกับคำสั่งเงื่อนไข if – else ได้ดังนี้

ตัวอย่างที่ 4 การใช้ตัวดำเนินการเทอร์นารี่

```
#include<stdio.h>
void main(){
    int x = 10 , y = 20 ,value_num;
    value_num = (x < y) ? x : y ;
    printf("value_num = %d",value_num);
}
```

เมื่อสั่งโปรแกรมทำงานผลโปรแกรมตัวอย่างที่ 4 จะปรากฏผลการทำงานดังต่อไปนี้

```
value_num = 10
```

จากตัวอย่างที่ 4 บรรทัด 4 จะเห็นว่ามีการใช้ตัวดำเนินการเทอร์นารี่ (x < y) ? x : y ; หมายถึง ถ้า x < y เป็นจริง ผลลัพธ์จะเป็นค่า x แต่ถ้า ถ้า x < y เป็นเท็จ ผลลัพธ์จะเป็นค่า y จากโค้ดจะเห็นว่าบรรทัดที่ 3 มีการกำหนดค่า x=10 และ y=20 เมื่อนำไปแทนค่า x < y จะได้ 10 < 20 เป็นจริง ดังนั้นผลลัพธ์ของตัวดำเนินการเทอร์นารี่จะเป็นค่า x นั้นคือ 10 จะถูกนำไปเก็บไว้ที่ตัวแปร value_num

5.2 ค่าคงที่ (Constant)

5.2.1 ความหมายของค่าคงที่ คือ ค่าของข้อมูลที่มีจำนวนแน่นอนเป็นค่าที่ไม่สามารถเปลี่ยนแปลงได้ขณะรันโปรแกรม ค่าคงที่แบ่งออกเป็น 3 ประเภท ดังนี้

1) ค่าคงที่ที่กำหนดโดยไม่ต้องมีตัวแปรรองรับ (Literal Constant) คือ ค่าคงที่ซึ่งเป็นข้อมูลที่แน่นอน ค่าคงที่ประเภทนี้ไม่จำเป็นต้องมีตัวแปรมารองรับสามารถกำหนดเข้าไปในโปรแกรมได้โดยตรงเลย เช่น

ตัวอย่างที่ 5 การใช้ค่าคงที่ที่กำหนดโดยไม่ต้องมีตัวแปรรองรับ

```
#include<stdio.h>
void main(){
    printf("3 x 3 = %d \n",9);
    printf("Hi ! %c Programming Language", 'C');
}
```

เมื่อสั่งโปรแกรมทำงานผลโปรแกรมตัวอย่างที่ 5 จะปรากฏผลการทำงานดังต่อไปนี้

```
3 x 3 = 9
Hi ! C Programming Language
```

2) ค่าคงที่ที่กำหนดที่ส่วนหัว (Header) ของโปรแกรม (Defined Constant) คือค่าคงที่ซึ่งกำหนดไว้ที่ส่วนหัวของโปรแกรมในลักษณะของ Preprocessing Directives (กำหนดไว้ก่อนฟังก์ชั่น main()) การกำหนดค่าคงที่ที่กำหนดที่ส่วนหัวมีรูปแบบดังนี้

#define ชื่อตัวแปร ค่าคงที่

ตัวอย่างที่ 6 การใช้ตัวดำเนินการเทอร์นารี่

```
#include<stdio.h>
#define PI 3.14159
void main(){
     printf("PI = %f ",PI);
}
```

เมื่อสั่งโปรแกรมทำงานผลโปรแกรมตัวอย่างที่ 6 จะปรากฏผลการทำงานดังต่อไปนี้

```
PI = 3.141590
```

3) ค่าคงที่ที่กำหนดให้กับตัวแปร (Memory Constant) คือ เมื่อทำการกำหนดค่าให้กับตัวแปรแล้ว ตัวแปรที่ถูกกำหนดค่าจะมีค่าเป็นเช่นนั้นไปตลอดการทำงานของโปรแกรม ไม่สามารถทำการเปลี่ยนแปลงค่าของตัวแปรนั้นได้ การกำหนดค่าคงที่ที่กำหนดให้กับตัวแปรมีรูปแบบดังนี้

const ชนิดของข้อมูล ชื่อค่าคงที่ = ค่า;

ตัวอย่างที่ 7 การใช้ตัวดำเนินการเทอร์นารี่

```
#include<stdio.h>
void main(){
     const float PI = 3.14159 ;
     printf("PI = %f ",PI);
}
```

เมื่อสั่งโปรแกรมทำงานผลโปรแกรมตัวอย่างที่ 7 จะปรากฏผลการทำงานดังต่อไปนี้

```
PI = 3.141590
```

5.2.2 การประกาศค่าคงที่แบ่งตามชนิดต่าง ๆ ได้ดังนี้

1) ค่าคงที่ชนิดจำนวนเต็ม (integer constant) เป็นค่าคงที่ชนิดตัวเลขซึ่งอาจจะเป็นค่าบวก 0 หรือ ค่าลบ ค่าคงที่ชนิดนี้มี 3 ประเภท คือ ค่าคงที่ integer ฐานสิบ ฐานแปดและฐานสิบหก ซึ่งมีรายละเอียดดังนี้

- ค่าคงที่ชนิดจำนวนเต็มฐานสิบ(decimal integer constant) หมายถึง ชุดของเลข ฐานสิบ (0 ถึง 9) ซึ่งเลขตัวแรกไม่เป็น 0 ดังตัวอย่างต่อไปนี้

```
const int price = 1000;
const int WeekDay = 7;
```

- ค่าคงที่ชนิดจำนวนเต็มฐานแปด (octal integer constant) หมายถึง ชุดของเลขฐานแปด (0 ถึง 7) ซึ่งเลขตัวแรกเป็น 0 ดังตัวอย่างต่อไปนี้

```
const int Oct1 = 01000;
const int Oct2 =037;
```

- ค่าคงที่ชนิดจำนวนเต็มสิบหก (hexadecimal integer constant) หมายถึง ชุดของเลขฐานสิบหก (0 ถึง 9 และ A ถึง F) ซึ่งนำหน้าด้วย 0X หรือ 0x ดังตัวอย่างต่อไปนี้

```
const int Hex1 = 0xF78;
const int Hex2 =0xFF;
```

2) ค่าคงที่ชนิดจำนวนจริง (floating point constant) เป็นค่าคงที่ชนิดตัวเลขที่มีทศนิยม ค่าคงที่ชนิดนี้มี 2 รูปแบบ ดังนี้

- เป็นชุดของตัวเลขที่ประกอบด้วยเลข 0 ถึง 9 และจุดทศนิยม ดังตัวอย่างต่อไปนี้

```
const float Pi = 3.1415;
const float InterestRate =0.65;
```

- เป็นเลขยกกำลัง ซึ่งมีส่วนประกอบ 2 ส่วนคือ ส่วนหนึ่งเป็น integer หรือ float อีกส่วนหนึ่งเป็นตัวคูณสิบยกกำลังซึ่งแสดงด้วยอักษร E (หรือ e) ตามด้วยเลขยกกำลังซึ่งเป็น integer บวกหรือลบดังตัวอย่างต่อไปนี้

const float A = 1.0E2; หมายถึง 1.0 * 10^2 ซึ่ง = 100.0

const float A = 1E+3; หมายถึง 1 * 10^3 ซึ่ง = 1000.0

const float A = 1E-2; หมายถึง 1 * 10^{-2} ซึ่ง = 0.01

3) ค่าคงที่ชนิดอักษร (character constant) เป็นค่าคงที่ที่ประกอบด้วยอักษรหนึ่งตัวอยู่ภายในเครื่องหมาย ‘ ’ ดังตัวอย่างต่อไปนี้

const char A = ‘a’;

const char B = ‘b’;

ค่าคงที่ชนิดอักษรมีชนิดเป็น char ค่าของค่าคงที่จะเป็นตัวเลขซึ่งมีค่าเท่ากับรหัส ASCII ของอักษรนั้น เช่น ‘A’= 65, ‘B’= 66

ค่าคงที่ชนิดอักษรอีกประเภทหนึ่งมีชื่อว่า เอสเคปซีเควนซ์ (escape sequence) ซึ่งเป็นค่าคงที่ที่ประกอบด้วยเครื่องหมาย \ และอักษรอยู่ภายในเครื่องหมาย (“ ”) เช่น “ \n ” , “ \t ” ค่าคงที่ประเภทนี้มีจำนวนจำกัดและมีความหมายเฉพาะดังแสดงในตารางที่ 5.8

ตาราง 5.8 เอสเคปซีเควนซ์

เอสเคปซีเควนซ์	ชื่อ	ความหมาย
\a	Alarm bell	เตรียมพร้อม (เสียงระฆัง)
\b	Backspace	เลื่อนกลับ
\f	Formfeed	ขึ้นหน้าใหม่
\n	Newline	ขึ้นบรรทัดใหม่
\r	Return	เคอเซอร์กลับไปอยู่ที่ต้นบรรทัด
\t	Tab	เว้นระยะในแนวระดับ
\\	Backslash	การกดเครื่องหมาย \\
\’	Single quote	การกดเครื่องหมาย \
\”	Double quote	การกดเครื่องหมาย “
\?	Question mark	การกดเครื่องหมาย ?

5.3 การเปลี่ยนแปลงชนิดของตัวแปร

ถ้านำตัวแปรต่างชนิดมาดำเนินการร่วมกัน ยกตัวอย่างเช่น int + float หรือ int – char การที่จะด ๆ เนินการตามเครื่องหมายได้นั้น จะต้องเปลี่ยนชนิดของตัวแปรให้เป็นชนิดเดียวกันก่อนโดย วิธีการเปลี่ยนชนิดของตัวแปรให้ตัวแปรในภาษา C ซึ่งมีอยู่ 2 รูปแบบด้วยกัน คือ เปลี่ยนโดยอัตโนมัติและเปลี่ยนโดยคำสั่งเปลี่ยนชนิดของตัวแปรโดยอัตโนมัติ

5.3.1 วิธีเปลี่ยนชนิดของตัวแปรโดยอัตโนมัติ จะเรียกว่า Implicit casting ผู้เขียนโปรแกรมไม่ต้องทำอะไร ตัวแปรภาษา C จะจัดการให้ทั้งหมด โดยใช้หลักการเปลี่ยนชนิดของตัวแปรที่ขนาดเล็กกว่าไปตามชนิดของตัวแปรที่มีขนาดใหญ่กว่าดังตารางที่ 5.9

ตาราง 5.9 ตัวอย่างการเปลี่ยนแปลงชนิดของตัวแปรอัตโนมัติ

ตัวดำเนินการที่ 1	ตัวดำเนินการที่ 2	ตัวดำเนินการที่ 3
char	int	char ---> int
int	long	int ---> long
int	unsigned int	int ---> unsigned int
int	float	int ---> float
int	double	int ---> double
float	double	float ---> double
long	double	long ---> double
double	long double	double ---> long double

5.3.2 วิธีเปลี่ยนชนิดของตัวแปรโดยคำสั่ง

การเปลี่ยนชนิดของตัวแปรโดยคำสั่งจะเรียกว่า Explicit casting เป็นการใช้คำสั่งเพื่อเปลี่ยนชนิดของตัวแปร สามารถเลือกชนิดของตัวแปรที่ต้องการจะเปลี่ยนไปใช้ได้ โดยรูปแบบคำสั่งแสดงดังนี้

```
(typecast)variable;
```

โดย typecast : ชนิดของตัวแปรที่ต้องการจะเปลี่ยนไปใช้

variable : ตัวแปรหรือข้อมูลที่ต้องการจะเปลี่ยนชนิด

ตัวอย่างที่ 8 วิธีเปลี่ยนชนิดของตัวแปรโดยคำสั่ง

```c
#include<stdio.h>
int x = 47;
float y = 8.25;
char z = 'A';
void main(){
    printf("y = %d\n",(int)y);
    printf("z = %d\n",(int)z);
    printf("%d\n",(int)-82.47);
    printf("%d\n",(int)'C');
    printf("%f\n",(float)x);
    printf("%f\n",(float)-125);
    printf("%f\n",(float)z);
}
```

เมื่อสั่งโปรแกรมทำงานผลโปรแกรมตัวอย่างที่ 8 จะปรากฏผลการทำงานดังต่อไปนี้

```
y = 8
z = 65
-82
67
47.000000
-125.000000
65.000000
```

5.4 การเปลี่ยนแปลงนิพจน์พีชคณิตเป็นนิพจน์ในภาษา C

สิ่งสำคัญสำหรับผู้เริ่มต้นที่จะเขียนโปรแกรม จำเป็นที่จะต้องทำความเข้าใจลำดับการทำงานของการประมวลผลของนิพจน์แล้ว ผู้เขียนโปรแกรมจะต้องเขียนนิพจน์หรือแปลงนิพจน์พีชคณิตให้มาเป็นนิพจน์คอมพิวเตอร์ในรูปภาษา C ให้ถูกต้องก่อนจึงจะสามารถประมวลผลได้ ดังแสดงการแปลงนิพจน์พีชคณิต ให้อยู่ในรูปแบบของภาษา C ที่สามารถประมวลผลได้ดังตัวอย่างต่อไปนี้

ตาราง 5.10 ตัวอย่างการเปลี่ยนแปลงนิพจน์พีชคณิตเป็นนิพจน์

นิพจน์ทางพีชคณิต	**นิพจน์พีชคณิตในภาษา C**
$\frac{(a^2 + b^2)}{(a - b)}$	((a * a) + (b * b)) / (a – b)
$\frac{a}{b} + \frac{c}{2d}$	a / b + c / (2 * d)
$\frac{a}{10 + 2\,(\frac{a}{a + b})}$	a / (10 + 2 * (a / (a + b)))

บทที่ 6

การรับและแสดงผลข้อมูล

6.1 ฟังก์ชันในการรับและแสดงผลข้อมูล

ในภาษา C การรับและการแสดงผลข้อมูลจะอาศัยการเรียกใช้ฟังก์ชันมาตรฐานมาทำงาน ซึ่งฟังก์ชันมาตรฐานเหล่านี้ แบ่งออกเป็น 2 ประเภท ได้แก่

1. ฟังก์ชันมาตรฐานสำหรับการรับและแสดงผลข้อมูลที่อยู่ใน ส่วนแรกสุดของไฟล์ (เฮดเดอร์ไฟล์) stdio.h (Standard I/O Library) ได้แก่ printf(); และ scanf();

2. ฟังก์ชันมาตรฐานสำหรับการรับและแสดงผลครั้งละหนึ่งตัวอักษรที่อยู่ในส่วนแรกสุดของไฟล์ (เฮดเดอร์ไฟล์) conio.h (Console I/O Library) ได้แก่ getchar(); , getche(); , getch(); ,gets(); , putchar(); และ puts();

6.2 ฟังก์ชันมาตรฐานสำหรับแสดงผล

6.2.1 ฟังก์ชัน printf();

ฟังก์ชันหรือคำสั่ง printf(); ซึ่งเป็นฟังก์ชันที่สามารถแสดงข้อมูลประเภทต่าง ๆ ออกทางหน้าจอภาพหรือแสดงผลออกทางอุปกรณ์เอาท์พุทอื่น ๆ ได้ เป็นคำสั่งที่มีความยืดหยุ่นเพราะสามารถกำหนดการจัดรูปแบบการแสดงผลให้มีระบียบและเหมาะสมตามต้องการ *เวลาจะใช้คำสั่งจะต้องบอกให้ตัวแปลชุดคำสั่งภาษาซีรับรู้ก่อน โดยใช้คำสั่ง #include <stdio.h> ไว้ส่วนหัวของโปรแกรม*

คำสั่ง printf(); สามารถแสดงผลค่าของตัวแปรออกทางหน้าจอได้ โดย จะต้องมีการใช้งานร่วมกับรหัสควบคุมรูปแบบเพื่อดึงค่าของตัวแปรมาแสดงผล ดังนั้นจะต้องเลือกใช้รหัสควบคุมรูปแบบให้ตรงกับชนิดตัวแปรหรือนิพจน์ที่ต้องการแสดงผล รหัสควบคุมรูปแบบแสดงได้ดังตาราง 6.1

ตาราง 6.1 รหัสควบคุมรูปแบบแสดงผลค่าของตัวแปร

รหัสควบคุม	ใช้สำหรับข้อมูลหรือตัวแปร
%c	อักขระ 1 ตัว (char)
%d	ตัวเลขจำนวนเต็ม (int, short, unsigned short, long, unsigned long)
%f	ตัวเลขทศนิยม (float, double, long double)
%s	ข้อความ (string หรืออักขระมากกว่า 1 ตัว)
%u	ตัวเลขจำนวนเต็มบวก (unsigned short, unsigned long)

ตาราง 6.1 รหัสควบคุมรูปแบบการรับและแสดงผลค่าของตัวแปร (ต่อ)

รหัสควบคุม	**ใช้สำหรับข้อมูลหรือตัวแปร**
%o	ข้อมูลแบบตัวเลขฐานแปด
%x	ข้อมูลแบบตัวเลขฐานสิบหก
%e	ตัวเลขทศนิยมในรูปแบบของ e ยกกำลัง (float, double, long double)
%E	ตัวเลขทศนิยมในรูปแบบของ E ยกกำลัง (float, double, long double)
%p	ตัวชี้ตำแหน่ง (pointer)
%%	ใช้แสดงเครื่องหมาย % 1 ตัว

คำสั่ง printf(); ยังสามารถจัดรูปแบบการแสดงผลให้ดูเป็นระเบียบมากขึ้น เช่น การขึ้นบรรทัดใหม่หลังแสดงข้อความ หรือเว้นระยะแท็บระหว่างข้อความ ในภาษา C มีรหัสควบคุมการแสดงผลหลายรูปแบบด้วยกัน แสดงดังตาราง 6.2

ตาราง 6.2 รหัสควบคุมการแสดงผลให้เป็นระเบียบและแสดงสัญลักษณ์พิเศษ

รหัสควบคุม	**ความหมาย**
\n	กำหนดให้ขึ้นบรรทัดใหม่
\t	กำหนดให้เว้นระยะในแนวนอน 1 แท็บ
\a	กำหนดเสียงกระดิ่ง (Bell)
\b	กำหนดให้ cursor ถอยหลังกลับไปทีละตัวพร้อมทั้งลบตัวอักษร ออก 1 ตัวอักษร หรือ ลดอักขระตัวสุดท้ายออก 1 ตัว
\r	กำหนดให้ cursor กลับไปอยู่ที่ต้นบรรทัด
\'	แสดงอักขระ ' 1 ตัว
\"	แสดงอักขระ " 1 ตัว
\\	แสดงอักขระ \ 1 ตัว

รูปแบบการใช้คำสั่ง printf มีทั้งหมด 3 รูปแบบ โดยข้อมูลที่ต้องการแสดงออกทางหน้าจอจะต้องเขียนไว้ในเครื่องหมาย “ ” รูปแบบใช้คำสั่ง printf แสดงได้ดังนี้

รูปแบบที่ 1 สำหรับการแสดงผลข้อความ

```
printf(“ข้อความ”);
```

ตัวอย่างที่ 1 การใช้ printf สำหรับการแสดงผลข้อความ

```
1 #include<stdio.h>
2 void main(){
3     printf("Hello C language!");
4 }
```

เมื่อสั่งรันโปรแกรมตัวอย่างที่ 1 จะปรากฏผลการทำงานดังต่อไปนี้

```
Hello C language!
```

จากตัวอย่าง 1 สามารถอธิบายส่วนประกอบที่สำคัญการทำงานดังนี้

บรรทัดที่ 1 จะใช้คำสั่ง printf(); จะต้องการเรียกอ่านแฟ้มส่วนหัว stdio.h ก่อน

บรรทัดที่ 2 ฟังก์ชันหลักของภาษาซี การเขียนโปรแกรมภาษาซีจะขาดฟังก์ชันนี้ไปไม่ได้ การทำงานของฟังก์ชันจะเริ่มต้นที่เครื่องหมาย { และจะสิ้นสุดที่เครื่องหมาย }

บรรทัดที่ 3 ใช้คำสั่ง printf(); เพื่อแสดงผลข้อความที่เขียนไว้ในเครื่องหมาย “ ” นั้นคือแสดงข้อความ Hello C language!

รูปแบบที่ 2 สำหรับการแสดงผลค่าของตัวแปร

```
printf(“รหัสควบคุม”, ตัวแปร);
```

ตัวอย่างที่ 2 การใช้ printf สำหรับการแสดงผลค่าของตัวแปร

```
1 #include<stdio.h>
2 void main(){
3     int x=5;
4     float y=0.23;
5     printf("%d \n %f", x,y);
6 }
```

เมื่อสั่งรันโปรแกรมตัวอย่างที่ 2 จะปรากฏผลการทำงานดังต่อไปนี้

```
5
 0.230000
```

จากตัวอย่าง 2 สามารถอธิบายส่วนประกอบที่สำคัญการทำงานดังนี้

บรรทัดที่ 3 ประกาศตัวแปร x เป็นชนิด int และกำหนดค่า x มีค่าเท่ากับ 5

บรรทัดที่ 4 ประกาศตัวแปร y เป็นชนิด float และกำหนดค่า y มีค่าเท่ากับ 0.23

บรรทัดที่ 5 ใช้คำสั่ง printf(); เพื่อแสดงผลค่าของตัวแปรที่กำหนดโดยจะแปลงค่ารหัสควบคุม และรูปแบบการแสดงผลที่อยู่ภายใน “ ” โดยจะจับคู่รหัสควบคุมกับตัวแปรตามลำดับจากซ้ายไปขวา ดังนั้น %d จะถูกแทนด้วยค่าของตัวแปร x นั้นคือ 5 แล้ว \n จะเป็นการขึ้นบรรทัดใหม่แล้วตามด้วย การเว้นวรรค 1 วรรค จากนั้นจะเปลี่ยน %f ด้วยค่าของตัวแปร y นั้นคือ 0.230000 ซึ่งโดยปกติการแสดงค่าของตำแหน่งทศนิยมโปรแกรมจะแสดงเป็นเลขทศนิยม 6 ตำแน่งเป็นค่าเริ่มต้น

รูปแบบที่ 3 สำหรับการแสดงผลข้อความและค่าของตัวแปร

```
printf("ข้อความ รหัสควบคุม", ตัวแปร);
```

ตัวอย่างที่ 3 การใช้ printf สำหรับการแสดงผลข้อความและค่าของตัวแปร

```
#include<stdio.h>
void main(){
    int x=5;
    float y=0.23;
        printf("x = %d \ny = %f",x,y);
}
```

เมื่อสั่งรันโปรแกรมตัวอย่างที่ 3 จะปรากฏผลการทำงานดังต่อไปนี้

```
x = 5
y = 0.230000
```

จากตัวอย่าง 3 สามารถอธิบายส่วนประกอบที่สำคัญการทำงานดังนี้

บรรทัดที่ 5 ใช้คำสั่ง printf เพื่อแสดงผลข้อความ แสดงผลค่าของตัวแปรที่กำหนดโดยจะแปลงค่ารหัสควบคุม และรูปแบบการแสดงผลที่อยู่ภายใน “ ” ดังนั้นจะแสดงข้อความ x = ตามด้วย 5 ที่เป็นการแทน %d ด้วยค่าของตัวแปร x จากนั้น \n จะเป็นการขึ้นบรรทัดใหม่แล้วตามด้วย แสดงข้อความ y = แล้วตามด้วย 0.230000 ซึ่งเป็นการแทน %f ด้วยค่าของตัวแปร y

1) การควบคุมขนาดพื้นที่แสดงผล

คำสั่ง printf สามารถกำหนดขนาดพื้นที่แสดงผลได้ ซึ่งตามปกติในการแสดงผลโปรแกรมจะเตรียมพื้นที่ให้พอดีกับข้อความ เช่น ถ้าจะแสดงข้อความ HELLO ซึ่งมี 5 ตัวอักษร โปรแกรมจะจัดเตรียมพื้นที่ไว้ให้พอดีสำหรับ 5 ตัวอักษร แสดงดังรูปที่ 6.1

H	E	L	L	O

รูปที่ 6.1 แสดงการจัดพื้นที่ในการแสดงผลแบบพอดีกับข้อความ

สามารถกำหนดขนาดพื้นที่ในการแสดงผลได้ตามต้องการ เช่น หากต้องการกำหนดให้แสดงข้อความ HELLO ในพื้นที่ขนาด 8 ตัวอักษร โปรแกรมจะแสดงข้อความชิดด้านขวาของพื้นที่ ที่จองไว้โดยจะเว้นพื้นที่ว่างทางด้านซ้ายอีก 3 ช่องที่เหลือเอาไว้ แสดงดังรูปที่ 6.2

			H	E	L	L	O

รูปที่ 6.2 แสดงจัดพื้นที่ในการแสดงผลแบบมากกว่าข้อความ

การกำหนดขนาดพื้นที่ในการแสดงผลนั้นให้ใส่ตัวเลขขนาดของพื้นที่ที่ต้องการไว้ข้างหลังเครื่องหมาย % และหากต้องการเปลี่ยนจากการแสดงผลชิดขอบด้านขวาให้ชิดขอบด้านซ้าย ให้ใส่เครื่องหมาย - ไว้หน้าตัว

เลขที่ใช้ระบุขนาดพื้นที่ในรหัสควบคุมรูปแบบ ถ้ากำหนดพื้นที่น้อยกว่าข้อมูลการแสดงผลการจะแสดงข้อมูลทั้งหมด แต่ถ้ากำหนดพื้นที่มากเกินไปจะมีช่องว่างให้เห็นในการแสดงผล ดังแสดงตัวอย่างต่อไปนี้

ตัวอย่างที่ 4 การกำหนดขนาดพื้นที่ในการแสดงผลผลลัพธ์ที่แสดงที่หน้าจอ

```
#include<stdio.h>
void main(){
    printf("|%s|\n", "HELLO");
    printf("|%10s|\n", "HELLO");
    printf("|%-10s|\n", "HELLO");
    printf("|%d|\n", 123);
    printf("|%2d|\n", 123);
    printf("|%-5d|\n", 123);
}
```

เมื่อสั่งรันโปรแกรมตัวอย่างที่ 4 จะปรากฏผลการทำงานดังต่อไปนี้

```
|HELLO|
|     HELLO|
|HELLO     |
|123|
|123|
|123  |
```

2) การควบคุมตำแหน่งตัวเลขหลังจุดทศนิยม

ในการแสดงผลตัวเลขหลังจุดทศนิยม ตามปกติถ้าไม่ได้กำหนดค่าใด ๆ เพิ่มเติม เมื่อใช้รหัสควบคุมรูปแบบ %f โปรแกรมจะแสดงตัวเลขทศนิยมทั้งหมด 6 ตำแหน่ง ในกรณีที่ต้องการกำหนดให้แสดงตัวเลขหลังจุดทศนิยมตามจำนวนที่ต้องการสามารถกำหนดค่าเพิ่มไปกับรหัสควบคุมรูปแบบได้ เช่น

%.nf คือการแสดงผลออกมาเป็นทศนิยมที่มีตัวเลขหลังจุด n ตัว เช่น %.3f จะแสดงตัวเลขหลังจุดทศนิยม 3 หลัก อย่างเช่น 12.345

%.ne คือการแสดงวผลออกมาในรูปทศนิยมเลขยกกำลังในรูป e และมีตัวเลขหลังจุดทศนิยม n ตัว เช่น %.3e จะแสดงตัวหลังเลขจุดทศนิยม 3 หลัก อย่างเช่น 1.256e + 02

%.nE คือการแสดงวผลออกมาในรูปทศนิยมเลขยกกำลังในรูป E และมีตัวเลขหลังจุดทศนิยม n ตัว เช่น %.3e จะแสดงตัวหลังเลขจุดทศนิยม 3 หลัก อย่างเช่น 1.256E + 02

นอกจากนี้ ถ้ากำหนดการควบคุมขนาดพื้นที่แสดงผลร่วมกับการควบคุมตำแหน่งตัวเลขหลังจุดทศนิยม พื้นที่การแสดงผลจะรวม ตัวเลขทั้งหมดและเครื่องหมายจุดด้วย ยกตัวอย่างเช่น 1234.56 ถ้าพื้นที่ที่พอดีคือ 7

ตัวอย่างที่ 5 การควบคุมตำแหน่งตัวเลขหลังจุดทศนิยม

```
#include<stdio.h>
void main(){
    printf("%e\n", 1234.56);
    printf("%E\n", 1234.56);
    printf("%f\n", 1234.56);
    printf("|%4.2f|\n", 1234.56);
    printf("|%3.1f|\n", 1234.56);
    printf("|%7.3f|\n", 1234.56);
    printf("|%8.3f|\n", 1234.56);
    printf("|%9.3f|\n", 1234.56);
}
```

เมื่อสั่งรันโปรแกรมตัวอย่างที่ 5 จะปรากฏผลการทำงานดังต่อไปนี้

```
1.234560e+003
1.234560E+003
1234.560000
|1234.56|
|1234.6|
|1234.560|
|1234.560|
| 1234.560|
```

3) การใช้รหัสควบคุมการแสดงผลเลขฐานแปดและเลขฐานสิบหก

โดยปกติในภาษาซีสามารถกำหนดค่าให้ตัวแปรเป็นเลขฐานต่าง ๆ ได้โดย

- เมื่อต้องการกำหนดค่าคงที่ชนิดจำนวนเต็มฐานสิบให้ตัวแปร ต้องใส่เลขตัวแรกไม่เป็น 0 เช่น 25, 3456 เป็นต้น
- เมื่อต้องการกำหนดค่าคงที่ชนิดจำนวนเต็มฐานแปดให้ตัวแปรให้ ต้องใส่เลขตัวแรกเป็น 0 เช่น 050, 0367 เป็นต้น

- เมื่อต้องการกำหนดค่าคงที่ชนิดจำนวนเต็มฐานสิบหกให้ตัวแปรให้ ต้องใส่เลขตัวแรกเป็น 0X หรือ 0x เช่น 0x65, 0X123 เป็นต้น

ถ้าหากต้องการแสดงค่าของตัวแปรเป็นเลขฐานต่าง ๆ สามารถใช้รหัสควบคุม ในการกำหนดการแสดงค่าเป็นเลขฐานต่าง ๆ ดังนี้

- %d เพื่อแสดงข้อมูลเป็นแบบตัวเลขฐานสิบ
- %o เพื่อแสดงข้อมูลเป็นแบบตัวเลขฐานแปด
- %x เพื่อแสดงข้อมูลข้อมูลแบบตัวเลขฐานสิบหก

ตัวอย่างที่ 6 การใช้รหัสควบคุมการแสดงผลเลขฐานแปดและเลขฐานสิบหก

```c
#include<stdio.h>
void main(){
    int x=165;
    printf("%d\n", x);
    printf("%o\n", x);
    printf("%x\n", x);
    printf("%X\n", x);
}
```

เมื่อสั่งรันโปรแกรมตัวอย่างที่ 6 จะปรากฏผลการทำงานดังต่อไปนี้

```
165
245
a5
A5
```

6.2.2 ฟังก์ชัน putchar();

ในการแสดงผลตัวอักษร 1 ตัว หรือ อักขระ 1 ตัว (char) ออกทางหน้าจอ นอกจากใช้ฟังก์ชัน printf พร้อมกับกำหนดรหัสควบคุมรูปแบบ %c แล้ว สามารถเรียกใช้คำสั่งสำหรับแสดงตัวอักษรหรืออักขระโดยเฉพาะได้อีกด้วย โดยใช้ฟังก์ชัน putchar( ); เป็นฟังก์ชันที่ใช้พิมพ์ข้อมูลประเภทตัวอักขระออกทางจอภาพโดยไม่มีการจัดรูปแบบ *เวลาจะใช้คำสั่งจะต้องบอกให้ตัวแปลชุดคำสั่งภาษาซีรับรู้ก่อน โดยใช้คำสั่ง #include <stdio.h> ไว้ส่วนหัวของโปรแกรม* มีรูปแบบดังนี้

```
putchar(ตัวแปรชนิดตัวอักขระหรือข้อมูลตัวอักขระ);
```

ตัวอย่างที่ 7 ใช้ฟังก์ชัน putchar(); เพื่อแสดงตัวอักษร

```c
#include<stdio.h>
void main(){
     char a;
     putchar('M');
     a = 'W';
     printf("\n");
     putchar(a);
}
```

เมื่อสั่งรันโปรแกรมตัวอย่างที่ 7 จะปรากฏผลการทำงานดังต่อไปนี้

```
M
W
```

6.2.3 ฟังก์ชัน puts();

ฟังก์ชัน puts(); เป็นฟังก์ชันที่ใช้สำหรับแสดงผลการพิมพ์ข้อความออกแสดงทางจอภาพ *เวลาจะใช้คำสั่งจะต้องบอกให้ตัวแปลชุดคำสั่งภาษาซีรับรู้ก่อน โดยใช้คำสั่ง #include <stdio.h> ไว้ส่วนหัวของโปรแกรม* มีรูปแบบดังนี้

```
puts(ค่าคงที่ชนิดสตริง);
```

ตัวอย่างที่ 8 ใช้ฟังก์ชัน puts(); ข้อความออกแสดงทางจอภาพ

```c
#include<stdio.h>
void main(){
     puts("Computer Program");
     puts("is Your Subject name.");
}
```

เมื่อสั่งรันโปรแกรมตัวอย่างที่ 8 จะปรากฏผลการทำงานดังต่อไปนี้

```
Computer Program
is Your Subject name.
```

6.3 ฟังก์ชันมาตรฐานในการรับข้อมูลจากคีย์บอร์ด

ในการทำงานของโปรแกรมส่วนใหญ่มักจะเป็นการเชื่อมโยงกับผู้ใช้แบบ 2 ทิศทาง นั่นก็คือ ทิศทางการแสดงผลการทำงานออกทางหน้าจอและทิศทางการรับข้อมูลจากผู้ใช้เข้ามาทางคีย์บอร์ด เพื่อร่วมในการประมวลผลของโปรแกรม ซึ่งในทิศทางของการรับข้อมูลจากผู้ใช้ ภาษา C จะกำหนดคำสั่งและฟังก์ชันมาตรฐานเอาไว้ให้เรียกใช้ เช่นเดียวกับทิศทางของการแสดงผล รายละเอียดของฟังก์ชันการรับข้อมูลจากคีย์บอร์ดได้แก่

6.3.1 ฟังก์ชัน scanf();

ในภาษา C การรับข้อมูลจากคีย์บอร์ดสามารถทำได้โดยการเรียกใช้ฟังก์ชัน scanf(); ซึ่งเป็นฟังก์ชันมาตรฐานสำหรับรับข้อมูลจากคีย์บอร์ด โดยสามารถรับข้อมูลได้ทุกประเภท ไม่ว่าจะเป็น จำนวนเต็ม (int), ทศนิยม (float), อักขระ (char) หรือข้อความก็ตาม รูปแบบการเรียกใช้คำสั่ง scanf(); ซึ่งเป็นฟังก์ชันที่ใช้ในการรับข้อมูลจากคีย์บอร์ดเข้าไปเก็บไว้ในตัวแปรที่กำหนดไว้ *จะถูกนิยามไว้ใน stdio.h* มีรูปแบบดังนี้

```
scanf("รหัสควบคุม", &ตัวแปร);
```

การใช้ฟังก์ชัน scanf(); จะต้องกำหนดโดยรหัสควบคุม (สามารถดูรหัสควบคุมได้ในตาราง 6.1) ให้ตรงกับชนิดของตัวแปรที่ใช้เก็บค่าที่รับจากคีย์บอร์ด และจะต้องใส่เครื่องหมาย & ไว้ด้านหน้าตัวแปร ยกเว้นตัวแปรเป็นสตริงสำหรับเก็บข้อความเท่านั้นที่ไม่ต้องนำหน้าด้วย เครื่องหมาย &

ตัวอย่างที่ 9 ใช้ฟังก์ชัน scanf(); รับข้อมูลจากคีย์บอร์ด

```
#include<stdio.h>
void main(){
     int x;
     char ch;
     char name[25];
     scanf(" %s", name);
     scanf(" %c", &ch);
     scanf(" %d", &x);
     printf("%c %s %d" , ch , name , x);
}
```

เมื่อสั่งรันโปรแกรมตัวอย่างที่ 9 จะปรากฏผลการทำงานดังต่อไปนี้ (ข้อความหรือตัวเลขที่ขีดเส้นใต้คือการรับข้อมูลจากผู้ใช้งานโปรแกรมผ่านแป้นพิมพ์)

```
Somjai
C
12345
C Somjai 12345
```

จากโปรแกรมหมายความว่า เครื่องจะรอรับค่าที่เข้ามา 3 ค่า คือ ค่าที่ 1 เป็นอักขระจำนวนไม่เกิน 25 ตัวขระ หรือ ตัวอักษรขนาด 25 ตัวอักษร ที่จะป้อนเข้ามาทางแป้นพิมพ์มาเก็บไว้ในตัวแปร name ส่วนค่าที่ 2 รับค่าตัวอักขระ 1 ตัวอักขระ ที่จะป้อนเข้ามาทางแป้นพิมพ์ มาเก็บไว้ในตัวแปร ch และ ส่วนค่าที่ 3 รับค่าเลขจำนวนเต็ม 1 ค่า ที่จะป้อนเข้ามาทางแป้นพิมพ์มาเก็บไว้ในตัวแปร x

6.3.2 ฟังก์ชัน getchar();

ฟังก์ชัน getchar (); เป็นฟังก์ชันที่ใช้ในการรับข้อมูลจากแป้นพิมพ์ทีละ 1 ตัวอักขระ โดยต้องกด enter ทุกครั้งเมื่อสิ้นสุดข้อมูล และข้อมูลที่ป้อนจะปรากฏให้เห็นบนหน้าจอภาพ ซึ่งเป็นฟังก์ชันที่ใช้ในการรับข้อมูลจากคีย์บอร์ดเข้าไปเก็บไว้ในตัวแปรที่กำหนดไว้ จะถูกนิยามไว้ใน conio.h มีรูปแบบดังนี้

```
ตัวแปร = getchar();
```

ตัวอย่างที่ 10 ใช้ฟังก์ชัน getchar (); รับข้อมูลจากคีย์บอร์ด 1 ตัวอักขระ

```
#include<stdio.h>
void main(){
    char ch;
    ch = getchar();
    printf("You typed : %c",ch);
}
```

เมื่อสั่งรันโปรแกรมตัวอย่างที่ 10 จะปรากฏผลการทำงานดังต่อไปนี้ (ข้อความหรือตัวเลขที่ขีดเส้นใต้คือการรับข้อมูลจากผู้ใช้งานโปรแกรมผ่านแป้นพิมพ์)

```
c
You typed : c
```

6.3.3 ฟังก์ชัน getch();

ฟังก์ชัน getch(); เป็นฟังก์ชันที่ใช้ในการรับข้อมูลเป็นตัวอักขระ 1 ตัวอักขระเข้ามาทางแป้นพิมพ์โดยเมื่อป้อนข้อมูลเสร็จไม่ต้องกดปุ่ม enter และข้อมูลที่ป้อนจะไม่ปรากฏบนหน้าจอภาพ *ถูกนิยามไว้ใน conio.h* มีรูปแบบดังนี้

ตัวอย่างที่ 11 ใช้ฟังก์ชัน getch(); รับข้อมูลจากคีย์บอร์ด 1 ตัวอักขระโดยเมื่อป้อนข้อมูลเสร็จไม่ต้องกดปุ่ม enter และข้อมูลที่ป้อนจะไม่ปรากฏบนหน้าจอภาพ

```c
#include<stdio.h>
void main(){
    char ch;
    ch = getch();
    printf("You typed : %c",ch);

}
```

เมื่อสั่งรันโปรแกรมตัวอย่างที่ 11 จะปรากฏผลการทำงานดังต่อไปนี้

```
You typed : A
```

6.3.4 ฟังก์ชัน getche();

ฟังก์ชัน getche(); เป็นฟังก์ชันที่ใช้ในการรับข้อมูลเป็นตัวอักขระ 1 ตัวอักขระเข้ามาทางแป้นพิมพ์โดยเมื่อป้อนข้อมูลเสร็จไม่ต้องกดปุ่ม enter และข้อมูลที่ป้อนจะปรากฏบนหน้าจอภาพ ถูกนิยามไว้ใน conio.h มีรูปแบบดังนี้

```
ตัวแปร = getche();
```

ตัวอย่างที่ 12 ใช้ฟังก์ชัน getche(); รับข้อมูลจากคีย์บอร์ด 1 ตัวอักขระโดยเมื่อป้อนข้อมูลเสร็จไม่ต้องกดปุ่ม enter

```
#include<stdio.h>
void main(){
    char ch ;
    printf("Enter a character : ");
    ch = getche();
    printf("\n ASCII code is %d",ch);
}
```

เมื่อสั่งรันโปรแกรมตัวอย่างที่ 12 จะปรากฏผลการทำงานดังต่อไปนี้ (ข้อความหรือตัวเลขที่ขีดเส้นใต้คือการรับข้อมูลจากผู้ใช้งานโปรแกรมผ่านแป้นพิมพ์)

Enter a character : <u>A</u>

ASCII code is 65

6.3.5 ฟังก์ชัน gets();

ฟังก์ชัน gets(); เป็นฟังก์ชันที่ใช้รับข้อมูลชนิดข้อความที่สามารถเว้นวรรคได้ (string) จากคีย์บอร์ด ถูกนิยามไว้ใน stdio.h มีรูปแบบดังนี้

gets(ตัวแปร);

ตัวอย่างที่ 13 ใช้ฟังก์ชัน gets(); รับข้อมูลที่เป็นข้อความจากคีย์บอร์ด

```
#include<stdio.h>
void main(){
    char ch ;
    char name[30];
    printf("Enter your name : ");
    gets(name);
    printf("\nYour name is : %s",name);
}
```

เมื่อสั่งรันโปรแกรมตัวอย่างที่ 13 จะปรากฏผลการทำงานดังต่อไปนี้ (ข้อความหรือตัวเลขที่ขีดเส้นใต้คือการรับข้อมูลจากผู้ใช้งานโปรแกรมผ่านแป้นพิมพ์)

```
Enter your name : somjai dodee

Your name is : somjai dodee
```

6.4 คำสั่งประกอบการควบคุมทิศทางแบบต่าง ๆ

6.4.1 คำสั่ง exit

คำสั่ง exit เป็นคำสั่งให้โปรแกรมหยุดทำงานทันที ใช้เพื่อออกจากการทำงานของโปรแกรมหลัก (Main Program) ในกรณีที่ผู้พัฒนาโปรแกรมต้องการให้หยุดการทำงานทั้งหมด มีรูปแบบดังนี้

```
exit(statuscode);
```

โดย statuscode เป็นตัวเลขที่บอกระบบว่าออกจากโปรแกรมโดยมีปัญหาหรือไม่ ปกติจะใส่ค่า 0 เพื่อแสดงว่าโปรแกรมไม่มีปัญหาในการสั่งจบโปรแกรม แต่ถ้าเป็นค่าอื่นหมายถึงนักเขียนโปรแกรมต้องการจะบอกกับระบบว่ามีปัญหาในการจบโปรแกรม ซึ่งก่อนจบงานคำสั่งนี้จะสั่งให้ปิดแฟ้ม ล้างบัฟเฟอร์

ตัวอย่างที่ 14 ใช้คำสั่ง exit เพื่อหยุดการทำงานของโปรแกรม

```c
#include<stdio.h>
void main(){
    printf("Before the Program is terminated. \n");
    exit(0);
    printf("This line is not never printed. \n");
}
```

เมื่อสั่งรันโปรแกรมตัวอย่างที่ 14 จะปรากฏผลการทำงานดังต่อไปนี้

```
Before the Program is terminated.
```

จากผลการรันตัวอย่างที่ 14 จะเห็นว่าผลการทำงานของโปรแรกมแสดงแค่ข้อความ Before the Program is terminated. ที่เป็นการแสดงผลจากโค้ดบรรทัดที่ 3 แต่ไม่มีการแสดงข้อความ This line is not never printed. จากโค้ดบรรทัดที่ 5 เนื่องจากบรรทัดที่ 4 มีคำสั่ง exit ที่ใช้เพื่อให้โปรแกรมหยุดทำงาน ดังนั้นโปรแกรมจะหยุดการทำงานตั้งแต่บรรทัดที่ 4 จึงทำให้คำสั่งบรรทัด 5 ไม่ได้ทำงาน

6.4.2 คำสั่ง break

คำสั่ง break ถูกนำไปใช้ร่วมกับคำสั่งแบบเลือกทำหรือวนรอบ เพื่อสั่งให้โปรแกรมหยุดการทำงานของคำสั่งแบบเลือกทำหรือวนรอบที่กำลังทำงานอยู่แล้วมาทำคำสั่งที่ต่อจาก loop นั้น

ตัวอย่างที่ 15 ใช้คำสั่ง break เพื่อหยุดการทำงานของคำสั่งแบบวนรอบ

```
1  #include<stdio.h>
2  void main(){
3      int x;
4      for(x = 1 ; x <= 10 ; x++){
5          if(x == 5)
6              break;
7          printf("%d",x);
8      }
9      printf("\nOut of loop at x = %d\n",x);
10 }
```

เมื่อสั่งรันโปรแกรมตัวอย่างที่ 15 จะปรากฏผลการทำงานดังต่อไปนี้

```
1234
Out of loop at x = 5
```

จากโปรแกรมตัวอย่างที่ 15 ได้ใช้คำสั่ง for loop เป็นคำสั่งวนซ้ำเพื่อวนแสดงตัวเลขจาก 1-10 โดยมีตัวแปร x เป็นตัวแปรที่ใช้ในการดำเนินการวนซ้ำ โดยค่า x เริ่มต้นที่ 1 และเพิ่มขึ้นที่ละ 1 ในแต่ละรอบของการทำซ้ำภายในการทำงานซ้ำแต่ละรอบจะมีการเช็คเงื่อนไขว่าถ้า x มีค่าเท่ากับ 5 เมื่อใดให้หยุดการทำงานซ้ำทันที (break ในบรรทัด 6) จึงทำให้ผลลัพธ์จากการทำงานของโปรแกรมนั้นแสดงเพียง 1-4

6.4.3 คำสั่ง continue

คำสั่ง continue เป็นคำสั่งที่ทำงานตรงข้ามกับคำสั่ง break ซึ่งสามารถใช้ในคำสั่ง for, while, do...while ได้เช่นกัน เมื่อโปรแกรมทำงานมาถึงคำสั่ง continue จะหยุดการทำงานในการวนรอบปัจจุบัน แล้วไปเริ่มทำงานในรอบต่อไปใหม่ทันที

ตัวอย่างที่ 16 การใช้คำสั่ง continue เพื่อหยุดการทำงานในการวนรอบปัจจุบัน

```
1  #include<stdio.h>
2  void main(){
3      int x;
4      for(x = 1 ; x <= 10 ; x++){
5          if(x == 5)
6              continue;
7          printf("%d",x);
8      }
9      printf("\nOut of loop at x = %d\n",x);
10 }
```

เมื่อสั่งรันโปรแกรมตัวอย่างที่ 16 จะปรากฏผลการทำงานดังต่อไปนี้

```
1234678910
Out of loop at x = 11
```

จากโปรแกรมตัวอย่างที่ 16 เป็นโปรแกรมที่ใช้คำสั่ง for loop เป็นคำสั่งวนซ้ำเพื่อวนแสดงตัวเลขจาก 1-10 เหมือนโปรแกรมตัวอย่างที่ 15 แต่แตกต่างกันตรงบรรทัดที่ 6 ในตัวอย่างนี้ใช้คำสั่ง continue จึงทำให้ผลลัพธ์จากการทำงานของโปรแกรมนั้นแสดงเพียง 1234678910 ไม่มีการแสดงผล 5 เนื่องจากคำสั่ง continue จะหยุดการทำงานในการวนรอบที่ x มีค่าเท่ากับ 5 ดังนั้นคำสั่งบรรทัดที่ 7 ที่เป็นการแสดงผลค่า x ในแต่ละรอบในรอบนั้นจึงไม่ทำงานจึงทำให้ไม่ปรากฏเลข 5 ในผลการทำงาน

บทที่ 7
การเขียนโปรแกรมแบบโครงสร้างโดยใช้คำสั่งตัดสินใจ

ในการเขียนโปรแกรมคอมพิวเตอร์บางครั้งจะต้องเขียนให้โปรแกรมมีทางเลือก เพื่อที่จะให้ทำงานอย่างใดอย่างหนึ่งตามเงื่อนไขที่เราต้องการ โดยถ้าเงื่อนไขเป็นจริง (True) จะทำงานตามที่คำสั่งที่ต้องการ แต่ถ้าเงื่อนไขเป็นเท็จ (False) จะไม่ทำงานตามคำสั่งนั้น ซึ่งในบทนี้จะกล่าวถึงโครงสร้างการทำงานแบบมีทางเลือกโดยจะแบ่งการทำงานออกเป็น 4 ลักษณะ ดังนี้คือ

1. คำสั่ง if
2. คำสั่ง if…else
3. คำสั่ง if…else if หรือ Nested if
4. คำสั่ง switch … case

7.1 คำสั่ง if

คำสั่ง if เป็นคำสั่ง ใช้ในกรณีที่ต้องการตรวจสอบเงื่อนไขการทำงาน (จริง/เท็จ) เพื่อให้เลือกกระทำคำสั่งที่ต่อจากคำสั่ง If.. หรือไม่ หลังจากนั้นจึงจะไปกระทำตามในคำสั่งถัดไปที่ มีรูปแบบการใช้งาน ดังนี้

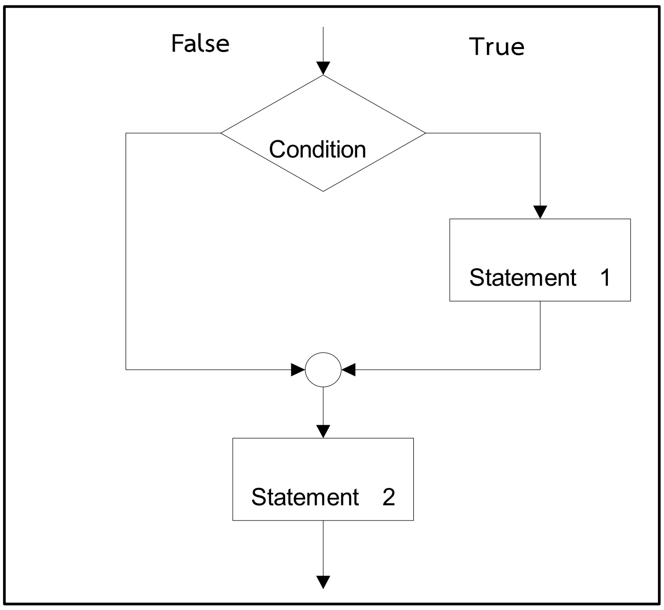
```
if (condition) statement;
```

- if ใช้ในกรณีที่มีทางเลือกอยู่เพียงทางเลือกเดียวคือทำหรือไม่ทำ
- condition : เงื่อนไขที่ให้ผลลัพธ์เป็นจริง หรือ เท็จ
- statement : คำสั่งที่จะให้ทำหากผลลัพธ์ของเงื่อนไขเป็น จริง
- หากมีคำสั่งมากกว่าหนึ่งคำสั่งจะต้องมีเครื่องหมายปีกกา ({ }) ครอบคำสั่งไว้

รูปแบบการทำงานของคำสั่ง if ในกรณีที่มีคำสั่งต้องการกระทำมากกว่า 1 คำสั่ง

```
if (condition) {
        statement_1;
        statement_2;
        statement_3;
        ----------------
        statement_n;
        }
```

ลักษณะการทำงานของคำสั่ง if สามารถเขียนเป็นผังงานได้ดังนี้



รูปที่ 7.1 ผังงานแสดงการทำงานของคำสั่ง if

ตัวอย่างโปรแกรมที่ 7.1

```
#include <stdio.h>
main()
{
    int n;
    printf(" Enter Number n= ");
    scanf("%d",&n);
    if (n>0)
        printf("positive number");
}
```

รูปที่ 7.2 ตัวอย่างโปรแกรมที่ 7.1 แสดงการทำงานของคำสั่ง if

จากตัวอย่างโปรแกรมที่ 7.1 เมื่อทำการรันโปรแกรม และป้อนค่า 5 ให้กับตัวแปร n จะแสดงผลข้อความ positive number ออกที่หน้าจอภาพ แต่ถ้าค่าที่ป้อนเท่ากับ 0 หรือ น้อยกว่า 0 จะไม่มีข้อความแสดงออกที่จอภาพ นั่นคือคำสั่ง printf จะไม่ถูกกระทำ เนื่องจากเงื่อนไขของคำสั่ง if เป็นเท็จ

ตัวอย่างโปรแกรมที่ 7.2

```
#include <stdio.h>
main()
{
    int n;
    printf(" Enter Number n= ");
    scanf("%d",&n);
    if (n>0)
        printf("positive number");
    printf("Good Bye");
}
```

รูปที่ 7.3 ตัวอย่างโปรแกรมที่ 7.2 แสดงการทำงานของคำสั่ง if

จากตัวอย่างโปรแกรมที่ 7.2 เมื่อทำการรันโปรแกรม และป้อนค่า 5 ให้กับตัวแปร n จะแสดงผลข้อความ positive number ออกที่หน้าจอภาพ และตามด้วยข้อความว่า Good Bye แต่ถ้าค่าที่ป้อนเท่ากับ 0 หรือ น้อยกว่า 0 จะมีข้อเฉพาะความข้อความว่า Good Bye แสดงออกที่จอภาพ นั่นคือคำสั่ง printf(“positive number”); จะไม่ถูกกระทำเพียงคำสั่งเดียว ส่วนคำสั่ง printf(“Good Bye”); จะถูก

กระทำทุกครั้ง ไม่ว่าเงื่อนไขของคำสั่ง if จะเป็นจริงหรือเท็จ ก็ตาม หมายความว่า คำสั่ง if จะตรวจสอบเงื่อนไข ถ้าเงื่อนไขเป็นจริง จะกระทำตามคำสั่งที่เขียนต่อไว้เพียงแค่คำสั่งเดียวเท่านั้น ถ้าต้องการให้กระทำมากกว่า 1 คำสั่งต้องเขียนทุกคำสั่งไว้ภายในเครื่องหมาย { }

ตัวอย่างโปรแกรมที่ 7.3

```
#include <stdio.h>
main()
{
    int n;
    printf(" Enter Number n= ");
    scanf("%d",&n);
    if (n>0)
    {
        printf("positive number");
        printf("Good Bye");
    }
}
```

รูปที่ 7.4 ตัวอย่างโปรแกรมที่ 7.3 แสดงการทำงานของคำสั่ง if

จากตัวอย่างโปรแกรมที่ 7.3 เมื่อทำการรันโปรแกรม และป้อนค่า 5 ให้กับตัวแปร n จะแสดงผลข้อความ positive number ออกที่หน้าจอภาพ และตามด้วยข้อความว่า Good Bye แต่ถ้าค่าที่ป้อนเท่ากับ 0 หรือ น้อยกว่า 0 จะไม่มีข้อเฉพาะความข้อความใดๆ แสดงออกมาที่จอภาพเลย นั่นคือคำสั่ง printf ทั้ง 2 คำสั่งจะถูกกระทำเมื่อเงื่อนไข if เป็นจริงอย่างเดียวเท่านั้น เพราะทั้ง 2 คำสั่งถูกเขียนไว้ภายในเครื่องหมาย { }

7.2 คำสั่ง if..else

คำสั่ง if...else เป็นคำสั่งที่ใช้ในกรณีที่ต้องการตรวจสอบเงื่อนไขการทำงาน (จริง/เท็จ) โดยต้องการที่จะเลือกกระทำคำสั่งอย่างใดอย่างหนึ่งจาก 2 อย่าง โดยดูจากเงื่อนไข ถ้าเงื่อนไขเป็นจริง (True) จะกระทำตามคำสั่งที่ต่อจาก if แต่ถ้าเงื่อนไขเป็นเท็จ (False) จะกระทำตามคำสั่งที่ต่อจาก else โดยมีรูปแบบการใช้งานดังนี้

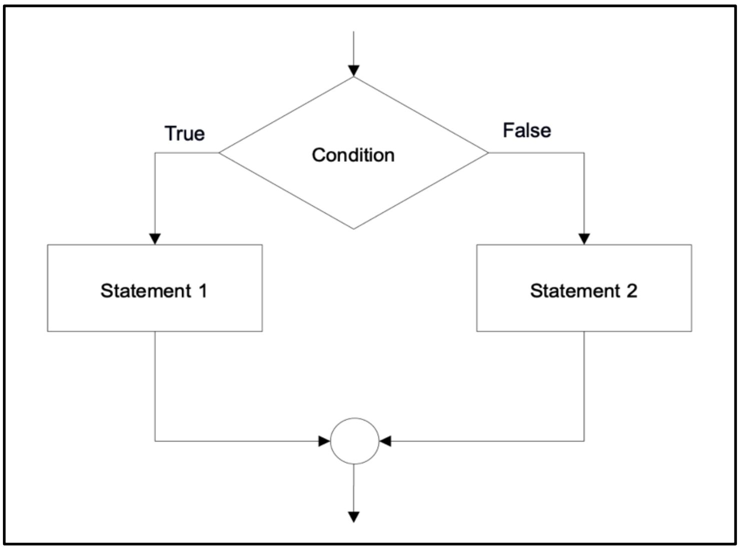
```
if (condition)
        statement A;
else
```

- if-else ใช้ในกรณีที่มีทางเลือกอยู่ 2 ทางเป็นจริงทำในส่วนของ if เป็นเท็จทำในส่วนของ else
- condition เป็นจริงจะทำ statement A หาก condition เป็นเท็จจะทำ statement B
- หากมีคำสั่งมากกว่าหนึ่งคำสั่งจะต้องมีเครื่องหมายปีกกา ({ }) ครอบคำสั่งไว้ทั้งในส่วนของ if และ else

รูปแบบการทำงานของคำสั่ง if..else ในกรณีที่มีคำสั่งต้องการกระทำมากกว่า 1 คำสั่ง

```
if (condition)
{
        statement A1;
        statement A2;
          ........................
        statement An;
}
else
{
        statement B1;
        statement B2;
          ........................
        statement Bn;
}
```

ลักษณะการทำงานของคำสั่ง if..else สามารถเขียนเป็นผังงานได้ดังนี้



รูปที่ 7.5 ผังงานแสดงการทำงานของคำสั่ง if..else

ตัวอย่างโปรแกรมที่ 7.4

```
#include<stdio.h>
main()
{
    int n;
    printf("Enter Integer Number : ");
    scanf("%d",&n);
    if(n>0)
        printf("positive number \n");
    else
        printf("negative number or zero\n");
}
```

รูปที่ 7.6 ตัวอย่างโปรแกรมที่ 7.4 แสดงการทำงานของคำสั่ง if..else

จากตัวอย่างโปรแกรมที่ 7.4 เมื่อทำการรันโปรแกรม และป้อนค่า 5 ให้กับตัวแปร n จะแสดงผลข้อความ positive number ออกที่หน้าจอภาพ แต่ถ้าค่าที่ป้อนเท่ากับ 0 หรือ น้อยกว่า 0 จะแสดงข้อความ negative number or zero ออกที่จอภาพ นั่นคือคำสั่ง if ตรวจสอบแล้วเงื่อนไขเป็นจริงจะกระทำคำสั่งที่ต่อจาก if แต่ถ้าเงื่อนไขเป็นเท็จจะไปกระทำในคำสั่งที่ต่อจาก else

ตัวอย่างโปรแกรมที่ 7.5

```
#include<stdio.h>
main()
{
    int n;
    printf("Enter Integer Number : ");
    scanf("%d",&n);
    if(n>0)
    {
        printf("positive number \n");
        printf("because %d > 0",n);
    }
    else
    {
        printf("negative number or zero\n");
        printf("because %d <= 0",n);
    }
    printf("Good Bye\n");
}
```

รูปที่ 7.7 ตัวอย่างโปรแกรมที่ 7.5 แสดงการทำงานของคำสั่ง if..else

จากตัวอย่างโปรแกรมที่ 7.5 เมื่อทำการรันโปรแกรม และป้อนค่า 5 ให้กับตัวแปร n จะแสดงผลข้อความ positive number , because 5 > 0 และตามด้วยข้อความ Good Bye ออกที่หน้าจอภาพ แต่ถ้าค่าที่ป้อนเท่ากับ 0 หรือ น้อยกว่า 0 เช่น -2 จะแสดงข้อความ negative number or zero , because -2 <= 0 และตามด้วยข้อความ Good Bye ออกที่จอภาพ จากผลการรันโปรแกรมสามารถอธิบายได้ว่า ถ้าเรามีคำสั่งที่ต้องการให้กระทำต่อจากการตรวจสอบเงื่อนไขในคำสั่ง if..else มากกว่า 1 คำสั่งจะต้องใส่ชุดคำสั่งเหล่านั้นไว้ภายในเครื่องหมาย { } ส่วนคำสั่งที่ไม่ได้อยู่ในเครื่องหมาย { } จะเป็นคำสั่งที่จะต้องถูกกระทำทุกครั้ง ไม่ว่าเงื่อนไขจะเป็นจริงหรือเท็จ ในตัวอย่างนี้ได้แก่คำสั่ง printf("Good Bye\n");

สรุปการใช้งานคำสั่ง if และ if..else ในการเลือกกระทำแบบต่างๆ มีข้อควรจำดังต่อไปนี้

- นิพจน์ (condition) ของ if จะต้องอยู่ในวงเล็บและให้ผลลัพธ์ออกมาเป็นจริงหรือเท็จ
- ในส่วนของ if และ else ไม่ต้องมีเครื่องหมาย semicolon (;)
- statement สามารถมีผลกระทบต่อนิพจน์เงื่อนไข (condition) ได้
- statement หลัง if และหลัง else สามารถเป็น statement รวมได้แต่ต้องอยู่ในเครื่องหมาย { }
- statement หลัง if และ statement หลัง else สามารถสลับกันได้แต่ต้องทำ complement กับนิพจน์เงื่อนไข (condition) ของ if

7.3 คำสั่ง if... else if หรือ Nested if

คำสั่ง if...else if หรือ Nested if จะเป็นโครงสร้างคำสั่งที่เพิ่มเข้ามาโดยใช้คำสั่ง if และ if..else ใช้ในกรณีที่มีทางเลือกที่ต้องการกระทำอยู่มากกว่า 2 ทางเลือก โดยแต่ละทางเลือกมีเงื่อนไขต่างกัน โดยลักษณะการใช้งานคำสั่งสามารถนำหลายๆ คำสั่งมาซ้อนกันได้ มีรูปแบบการใช้งาน ดังนี้

```
if(condition A)
   statement A;
else if(condition B)
   statement B;
else if(condition C)
   statement C;
............................
else
   statement n;
```

(a)

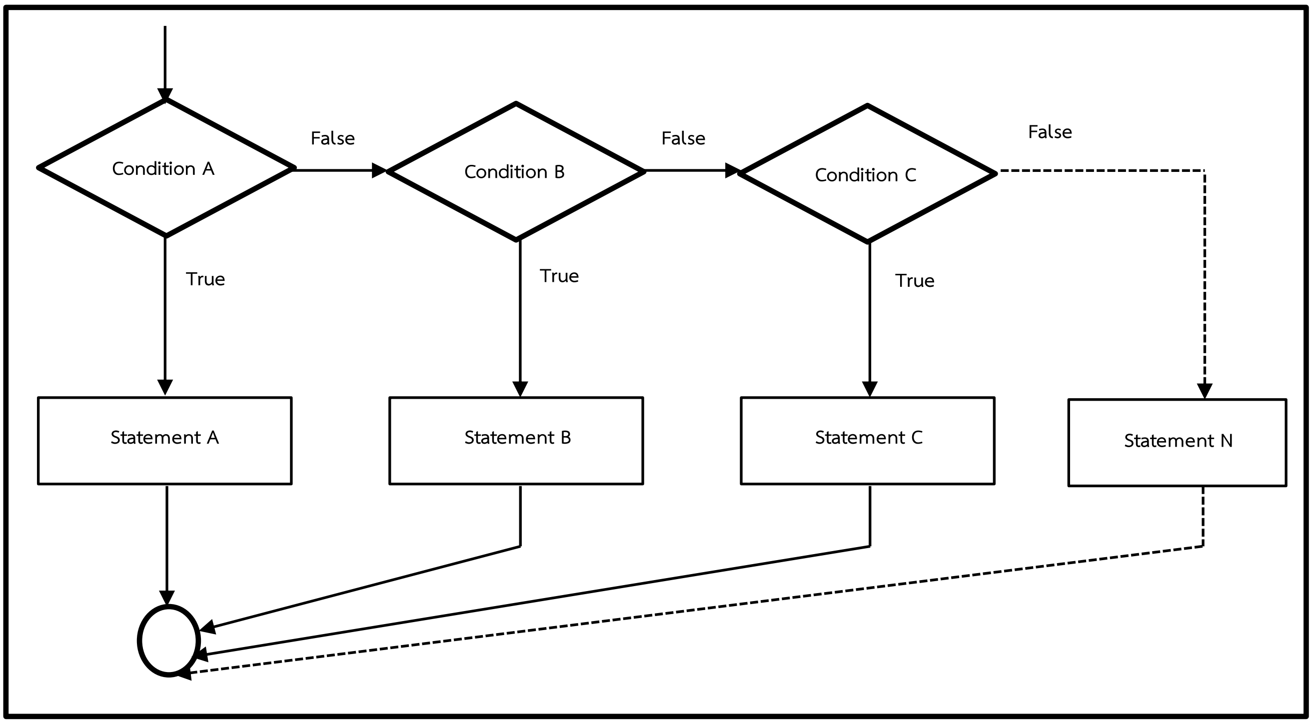
```
if(condition A)
{
   statement A1;
   ...
}
else if(condition B)
{
    statement B1;
   ...
}
else
{
   statement  n;
   ...
}
```

(b)

รูปที่ 7.8 (a) และ (b) แสดงรูปแบบการทำงานของคำสั่ง if..else..if หรือ Nested if

หมายเหตุ else ตัวสุดท้ายจะมีหรือไม่มีก็ได้ขึ้นอยู่กับเงื่อนไขที่ต้องการ

ลักษณะการทำงานของคำสั่ง if..else..if หรือ Nested if สามารถเขียนเป็นผังงานได้ดังนี้



รูปที่ 7.9 ผังงานแสดงการทำงานของคำสั่ง if..else..if หรือ Nested if

ตัวอย่างโปรแกรมที่ 7.6

```
#include <stdio.h>
int main()
{
  int a,b;
  printf("Please enter two integer: ");
  scanf("%d %d",&a,&b);
  if(a<=b)
    if(a<b)
       printf("%d < %d\n",a,b);
     else
      printf("%d = %d\n",a,b);
  else
     printf("%d > %d\n",a,b);
}
```

รูปที่ 7.10 ตัวอย่างโปรแกรมที่ 7.6 แสดงการทำงานของคำสั่ง if..else..if หรือ Nested if

จากตัวอย่างโปรแกรมที่ 7.6 เป็นตัวอย่างการทำงานของโปรแกรมเพื่อทำการตรวจสอบค่าตัวเลข 2 ค่าที่ป้อนเข้ามาระหว่างค่าในตัวแปร a และ b ว่ามีค่า > , < หรือ = ซึ่งจะพบว่ามีทางเลือกที่ต้องการผลลัพธ์ในการกระทำมากกว่า 2 ทางเลือก ในการเขียนคำสั่งในการตรวจสอบเงื่อนไขจึงจำเป็นต้องใช้โครงสร้างคำสั่งแบบ if..else..if ช่วย

ตัวอย่างโปรแกรมที่ 7.7

```
#include<stdio.h>
#include<conio.h>
main()
{
        int a,b;
        char ch;
        printf("Do you want to \n");
        printf("A:Add, S:Subtract, M: Multiply, or D:Divide ? \n ");
        printf("Enter the first letter :");
        ch=getchar();
        printf("Enter the first number :");
        scanf("%d",&a);
        printf("Enter the second number :");
        scanf("%d",&b);
        if ((ch=='A')||(ch=='a'))
                printf("result = %d",a+b);
        else if ((ch=='S')||(ch=='s'))
                        printf("result = %d",a-b);
                else if ((ch=='M')||(ch=='m'))
                                printf("result = %d",a*b);
                        else if ( ((ch=='D')||(ch=='d')) && (b!=0))
                                printf("result = %d",a/b);
                                else
                                        printf("ERROR");
}
```

รูปที่ 7.11 ตัวอย่างโปรแกรมที่ 7.7 แสดงการทำงานของคำสั่ง if..else..if หรือ Nested if

จากตัวอย่างโปรแกรมที่ 7.7 เป็นตัวอย่างการเขียนโปรแกรมโดยใช้ if..else..if หรือ Nested if เพื่อตรวจสอบเงื่อนไขที่เป็นตัวอักษร A หรือ a, S หรือ s, M หรือ m และ D หรือ d ในการเลือกกระทำการ บวก ,ลบ ,คูณ หรือ หาร ระหว่างค่าของตัวเลขที่อยู่ในตัวแปร a และ b โดยจะไล่ตรวจสอบเงื่อนไขตามลำดับของคำสั่ง if ที่เขียนไว้ ซึ่งถ้าป้อนค่าไม่ตรงกับเงื่อนไขใดเลยจะมาทำในส่วน else สุดท้าย แสดงข้อความ ERROR ออกมาที่จอภาพ

ตัวอย่างโปรแกรมที่ 7.8

```
#include <stdio.h>
main()
{
        int select,num;
        printf("Convert: \n");
        printf("1. Decimal to Hexadecimal\n");
        printf("2. Hexadecimal to Decimal\n");
        printf("3. Decimal to Octal\n");
        printf("4. Octal to Decimal\n");
        printf("Enter your choice : ");
        scanf("%d",&select);
        if(select==1 )
        {
                printf("Enter decimal number : ");
                scanf("%d",&num);
                printf("%d in hexadecimal is %X",num,num);
        }
        else if (select==2)
                {
                        printf("Enter hexadecimal number : ");
                        scanf("%X",&num);
                        printf("%X in decimal is %d",num,num); }
                else if (select==3)
                        {
                                printf("Enter decimal number : ");
                                scanf("%d",&num);
                                printf("%d in octal is %o",num,num);
                        }
                        else if (select==4)
                                {
                                        printf("Enter octal number : ");
                                        scanf("%o",&num);
                                        printf("%o in decimal is %d",num,num);
                                }

}
```

รูปที่ 7.12 ตัวอย่างโปรแกรมที่ 7.8 แสดงการทำงานของคำสั่ง if..else..if หรือ Nested if

จากตัวอย่างโปรแกรมที่ 7.8 จะมีโครงสร้างคล้ายกับโปรแกรมที่ 7.7 คือมีหลักการในการเขียนโปรแกรมโดยใช้โครงสร้างของการเขียนคำสั่งเพื่อการตรวจสอบเงื่อนไขเหมือนกัน ต่างกันในส่วนรายละเอียดของการรับค่าและการแสดงผลข้อมูล

ตัวอย่างโปรแกรมที่ 7.9

```
#include <stdio.h>
main()
{
    int mark;
    printf("Enter your point : ");
    scanf("%d",&mark);
    if((mark<=100)&&(mark>=80))
      printf("Grade A\n");
    else if((mark<80)&&(mark>=70))
              printf("Grade B\n");
         else if((mark<70)&&(mark>=60))
                  printf("Grade C\n");
              else if((mark<60)&&(mark>=50))
                       printf("Grade D\n");
                   else
                         printf("Grade F\n");
}
```

รูปที่ 7.13 ตัวอย่างโปรแกรมที่ 7.9 แสดงการทำงานของคำสั่ง if..else..if หรือ Nested if

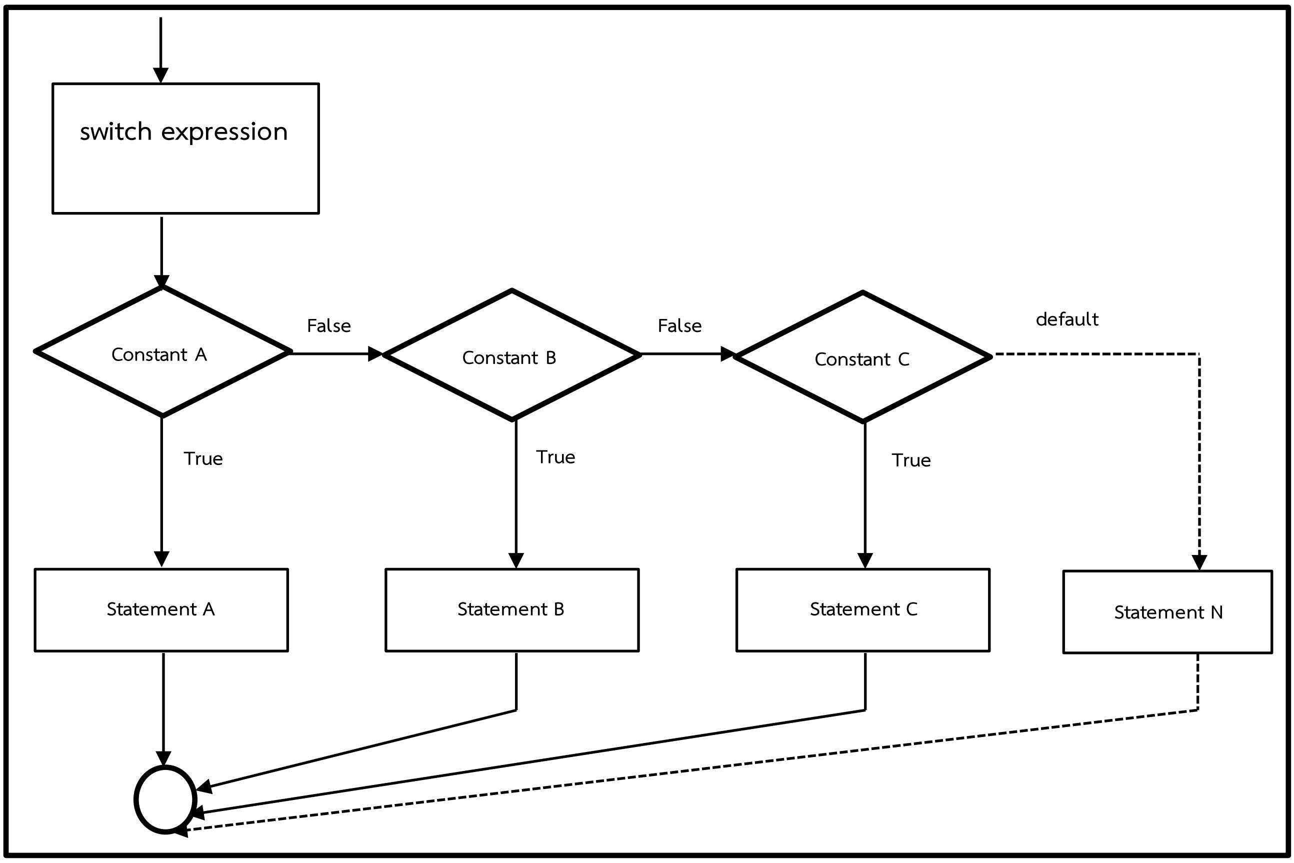
7.4 คำสั่ง switch ... case

คำสั่ง switch..case เป็นคำสั่งที่ใช้ในการตรวจสอบเงื่อนไขเพื่อเลือกกระทำคำสั่งที่มีทางเลือกของการตัดสินใจหลายทาง ซึ่งสามารถทำงานได้เช่นเดียวกับคำสั่ง if...else if หรือ Nested if แต่นิยมใช้คำสั่ง switch..case มากกว่าเพราะมีรูปแบบที่เข้าใจได้ง่าย สะดวกในการแก้ไขในกรณีที่มีการผิดพลาด อย่างไรก็ตามการเลือกใช้ switch..case ยังมีข้อจำกัดบางประการเนื่องจากจะตรวจสอบค่าในตัวแปร (variable) ที่เป็นลักษณะของค่าคงที่ (constant) และสามารถใช้งานได้กับชนิดข้อมูลที่เป็นประเภท int หรือ char เท่านั้น ดังนั้นในการพิจารณาเลือกใช้ if...else if หรือ switch..case จะต้องพิจารณาในส่วนนี้ด้วย มีรูปแบบดังนี้

```
switch(variable)
{
  case constant A : Statement A1;
                    Statement A2;
                    ...
                    break;
  case constant B : Statement B1;
                    Statement B2;
                    ...
                    break;
  ...
  default :
     statement;
```

- switch ใช้ในกรณีที่มีทางเลือกอยู่หลายทางเลือก
- variable : ตัวแปรหรือนิพจน์ที่ให้ผลเป็นข้อมูลแบบ int หรือ char
- constant A , constant B,... : ค่าคงที่ชนิดเดียวกับตัวแปร variable หากค่าของ variable เท่ากับค่าคงที่ตัวใด โปรแกรมจะทำคำสั่ง case นั้น
- break : ใช้คั่นระหว่าง case ให้โปรแกรมออกจากทำงานของ switch หากไม่ใส่ตัวคั่น โปรแกรมจะทำการเปรียบเทียบ Variable กับ constant ตัวถัดไปเรื่อย ๆ
- default : หาก Variable ไม่ตรงกับ constant ใดเลย โปรแกรมจะทำตามคำสั่งใน default

ลักษณะการทำงานของคำสั่ง if..else..if หรือ Nested if สามารถเขียนเป็นผังงานได้ดังนี้



รูปที่ 7.14 ผังงานแสดงการทำงานของคำสั่ง switch..case

สิ่งสำคัญในการใช้งานคำสั่ง switch..case

- คำสั่ง switch..case ต่างจากคำสั่ง if..else..if ตรงที่ switch..case สามารถตรวจสอบเงื่อนไขได้หลายอย่าง แต่คำสั่ง if..else..if จะตรวจสอบเฉพาะความสัมพันธ์ หรือลอจิกเท่านั้น
- ถ้าค่าคงที่เป็นตัวอักษร (char) คำสั่ง switch จะมองเป็นเลขจำนวนเต็ม (int)
- ค่า default จะมีหรือไม่มีก็ได้

ตัวอย่างโปรแกรมที่ 7.10

```
#include<stdio.h>
#include<conio.h>
main()
{
        char Operator;
        int ans,x,y;
        x=10; y=2;
        printf("Enter operator : ");
        Operator=getche();
        printf("\n");
        switch (Operator)
        {
                case '+' : ans =x + y;
                           printf("answer = %d",ans);
                           break;
                case '-' : ans =x - y;
                           printf("answer = %d",ans);
                           break;
                case '*' : ans =x * y;
                           printf("answer = %d",ans);
                           break;
                case '/' : ans = x / y;
                           printf("answer = %d",ans);
                           break;
                default  : printf("This program can use only +,-,*,/ \n");
        }
}
```

รูปที่ 7.15 ตัวอย่างโปรแกรมที่ 7.10 แสดงการทำงานของคำสั่ง switch..case

จากตัวอย่างโปรแกรมที่ 10 เป็นตัวอย่างของการเขียนโปรแกรมโดยใช้คำสั่ง switch..case ซึ่งกำหนดให้ตัวแปรที่ทำหน้าที่ในการเก็บค่าที่นำมาตรวจสอบคือ Operator จากตัวอย่างกำหนดให้เป็นชนิด char ดังนั้นในส่วนของค่าคงที่ (Constant) ต้องเขียนในรูปแบบเดียวกันกับชนิดข้อมูลด้วย คือ '+','-','*' และ '/' และกรณีที่ป้อนค่านอกเหนือจากเงื่อนไข หรือไม่ตรงกับ case ที่กำหนดจะไปทำงานในส่วน default

ตัวอย่างโปรแกรมที่ 7.11

```
#include <stdio.h>
main()
{
        int select,num;
        printf("Convert: \n");
        printf("1. Decimal to Hexadecimal\n");
        printf("2. Hexadecimal to Decimal\n");
        printf("3. Decimal to Octal\n");
        printf("4. Octal to Decimal\n");
        printf("Enter your choice : ");
        scanf("%d",&select);
        switch(select)
        {
                case 1 : printf("Enter decimal number :");
                         scanf("%d",&num);
                         printf("%d in hexadecimal is %X",num,num);
                         break;
                case 2 : printf("Enter hexadecimal number : ");
                         scanf("%X",&num);
                         printf("%X in decimal is %d",num,num);
                         break;
                case 3 : printf("Enter decimal number : ");
                         scanf("%d",&num);
                         printf("%d in decimal is %o",num,num);
                         break;
                case 4 : printf("Enter octal number : ");
                         scanf("%o",&num);
                         printf("%o in decimal is %d",num,num);
                         break;
                default : printf("Invalid selection , try again\n");
                          break;
        }
}
```

รูปที่ 7.16 ตัวอย่างโปรแกรมที่ 7.11 แสดงการทำงานของคำสั่ง switch..case

จากตัวอย่างโปรแกรมที่ 7.11 จะมีหลักการทำงานเดียวกันกับตัวอย่างโปรแกรมที่ 7.10 ต่างกันตรงที่โปรแกรมนี้กำหนดให้ตัวแปรที่ทำหน้าที่ในการเก็บค่าที่นำมาตรวจสอบคือ select จากตัวอย่างกำหนดให้เป็นชนิด int ดังนั้นในส่วนของค่าคงที่ (Constant) ต้องเขียนในรูปแบบเดียวกันกับชนิดข้อมูลด้วย คือ 1,2,3 หรือ 4

จากตัวอย่างโปรแกรมที่ 7.11 นี้ ยังสามารถนำวิธีการเขียนโปรแกรมไปเทียบกับตัวอย่างโปรแกรมที่ 7.8 เนื่องจากทั้ง 2 โปรแกรมได้รูปแบบผลลัพธ์การรันโปรแกรมที่เหมือนกันทุกประการ แต่ใช้โครงสร้างของคำสั่งในการเขียนโปรแกรมแตกต่างกัน ดังนั้นผู้เขียนโปรแกรมควรเลือกใช้คำสั่งที่เหมาะสมและให้ผลการ

ทำงานที่มีประสิทธิภาพมากที่สุด เมื่อเปรียบเทียบการทำงานของทั้ง 2 โปรแกรมนี้ ตัวอย่างโปรแกรมที่ 7.11 จะทำงานได้ดีที่สุด เพราะไม่ว่าจะทำการป้อนค่าใดๆ เข้าไปจำนวนของ case ที่ถูกตรวจสอบเงื่อนไขและกระทำเพียงแค่ case เดียว แต่จากตัวอย่างโปรแกรมที่ 7.8 ถ้าเราป้อนค่าให้กับตัวแปร select = 3 คำสั่ง if ที่ถูกตรวจสอบจะไล่มาจากที่คำสั่ง if ที่เขียนไว้อันดับแรก และไล่ตรวจสอบลงมาตามลำดับ ทำให้มีการทำงานที่ช้ากว่าตัวอย่างโปรแกรมที่ 7.11

แต่จากตัวอย่างโปรแกรมที่ 7.7 และ 7.9 เป็นการเขียนโดยใช้คำสั่ง if..else..if ซึ่งเขียนเงื่อนในการตรวจสอบค่าเป็นลอจิก เป็นค่าในการตรวจสอบที่มาก จะว่า 1 ค่า จึงไม่เหมาะสมที่จะนำการทำงานแบบนี้มาเขียนโดยใช้คำสั่ง switch..case

ดังนั้น จึงพบสรุปได้ว่าแต่ละโครงสร้างคำสั่งจะมีลักษณะเด่น และข้อจำกัดในการใช้งานแตกต่างกันไป ผู้เขียนโปรแกรมจะต้องเลือกใช้คำสั่งให้เหมาะสม เพื่อให้โปรแกรมทำงานได้มีประสิทธิภาพมากที่สุด

บทที่ 8
การเขียนโปรแกรมทำงานแบบวนรอบ

8.1 บทนำ

จากบทที่ผ่านมาจะเป็นการเขียนโปรแกรมให้ทำงานเรียงตามลำดับของคำสั่ง โดยการทำงานจะเริ่มต้นจากคำสั่งแรก คำสั่งที่สองไปจนถึงคำสั่งสุดท้ายตามลำดับ ถ้าโปรแกรมมีการทำงานที่เหมือนหรือซ้ำกันก็จะต้องเขียนกลุ่มคำสั่งซ้ำๆ กัน ซึ่งถ้าโปรแกรมมีการทำงานเหมือนหรือซ้ำกันหนึ่งร้อยครั้งก็จะต้องเขียนกลุ่มคำสั่งซ้ำกันหนึ่งร้อยครั้งทำให้โปรแกรมมีขนาดใหญ่และทำการแก้ไขยาก ปัญหานี้สามารถแก้ไขได้โดยใช้การเขียนโปรแกรมทำงานแบบวนรอบ

โปรแกรมทำงานแบบวนรอบ (Loop) คือ โปรแกรมที่สามารถควบคุมให้คำสั่งกลุ่มใดกลุ่มหนึ่งทำงานซ้ำๆกัน ตามจำนวนครั้งที่กำหนดหรือจนกว่าผลการตรวจสอบเงื่อนที่กำหนดจะเป็นเท็จ ซึ่งข้อดีของโปรแกรมทำงานแบบวนรอบจะช่วยให้เขียนและปรับแก้โปรแกรมได้ง่าย กระชับ มีความยืดหยุ่น และมีขนาดเล็ก โปรแกรมภาษาซีมีคำสั่งทำงานแบบวนรอบ 3 คำสั่งดังนี้

1. คำสั่งวนรอบ for
2. คำสั่งวนรอบ while
3. คำสั่งวนรอบ do...while

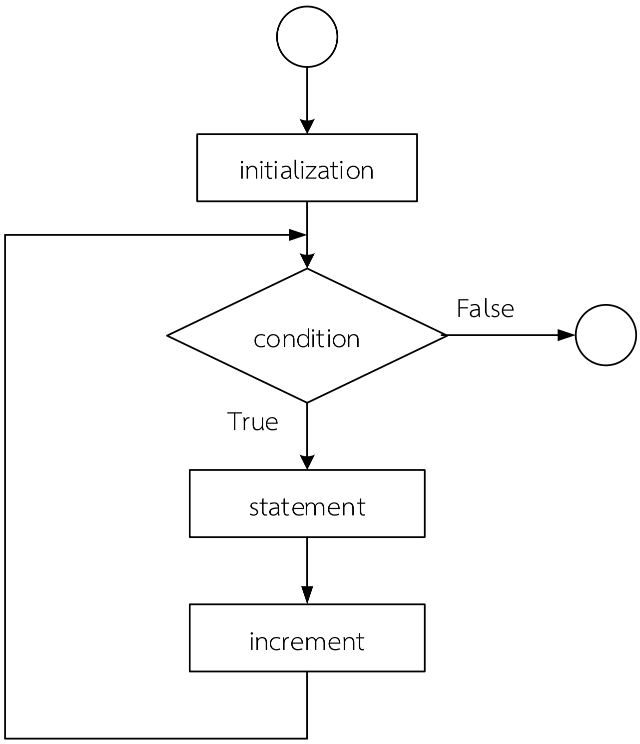
8.2 คำสั่งวนรอบ for

คำสั่งวนรอบ for เป็นคำสั่งที่ทำหน้าที่ควบคุมให้กลุ่มคำสั่งทำงานซ้ำตามจำนวนรอบที่กำหนด โดยจะมีการกำหนดค่าเริ่มต้นให้กับตัวแปรควบคุมลูปและเงื่อนไขการสิ้นสุดที่แน่นอน ซึ่งช่วยลดขนาดและเพิ่มความเร็วในการประมวลผลของโปรแกรม มีรูปแบบดังนี้

```
for(initialization; condition; increment)
{
        statement;
}
```

โดยที่	initialization	คือ นิพจน์กำหนดค่าเริ่มต้นให้กับตัวแปรควบคุมการวนรอบ
	condition	คือ นิพจน์เงื่อนไขมีค่าจริงหรือเท็จอย่างใดอย่างหนึ่งเท่านั้น
	increment	คือ นิพจน์เพิ่มหรือลดค่าตัวแปรควบคุมการวนรอบ
	statement	คือ กลุ่มคำสั่งที่กำหนดให้ทำงานซ้ำ เมื่อผลการตรวจสอบเงื่อนไขเป็นจริง

การทำงานของคำสั่งวนรอบ for จากแผนผังในรูปที่ 8.1 เมื่อคำสั่ง for เริ่มทำงาน ตัวแปรควบคุมการวนรอบจะถูกกำหนดให้มีค่าเท่ากับค่าเริ่มต้น จากนั้นเงื่อนไขจะถูกตรวจสอบ ถ้าผลการตรวจสอบเป็นจริงกลุ่มคำสั่งที่กำหนดให้ทำงานซ้ำจะทำงาน ต่อจากนั้นตัวแปรควบคุมการวนรอบจะถูกเพิ่มหรือลดค่า จากนั้นจะย้อนกลับไปตรวจสอบเงื่อนไขอีกครั้ง ถ้าผลการตรวจสอบเงื่อนไขเป็นจริงกลุ่มคำสั่งที่กำหนดให้ทำงานซ้ำจะทำงานอีกครั้ง แต่ถ้าผลการตรวจสอบเป็นเท็จการทำงานของคำสั่ง for จะสิ้นสุดลง และข้ามไปทำคำสั่งที่อยู่ถัดไป



รูปที่ 8.1 แผนผังแสดงการทำงานของคำสั่งวนรอบ for

ตัวอย่าง โปรแกรมแสดงเลข 1 ถึง 5 ออกหน้าจอภาพ โดยคำสั่งวนรอบ for

```
#include<stdio.h>
int main()
{
    int   i;
    for(i = 1 ; i <= 5 ; i++)
    {
        printf("%d\n",i);
    }
    return(0);
}
```

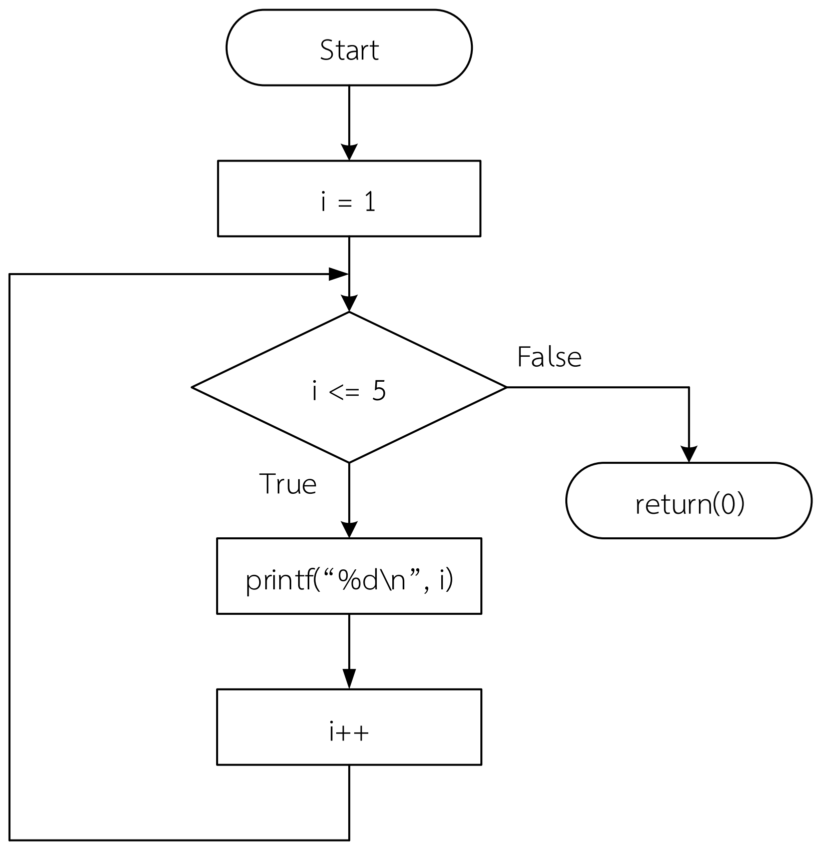
จากตัวอย่างโปรแกรมสามารถอธิบายการทำงานของคำสั่งบางส่วนได้ดังนี้

บรรทัดที่ 4 ประกาศให้ i เป็นตัวแปรที่มีชนิดข้อมูลเป็นเลขจำนวนเต็ม

บรรทัดที่ 5 คำสั่งวนรอบ for จะเริ่มต้นทำงานโดยกำหนดให้ตัวแปร i มีค่าเริ่มต้นเท่ากับ 1 (i = 1) จากนั้นทำการตรวจสอบเงื่อนไข i <= 5 ถ้าผลตรวจสอบเงื่อนไขเป็นจริง จะไปทำคำสั่งบรรทัดที่ 7 (printf) หลังจากทำคำสั่งบรรทัดที่ 7 เสร็จ จะทำการเพิ่มค่าตัวแปร i ขึ้นอีก 1 (i++) จากนั้นจะย้อนกลับไปทำการตรวจสอบเงื่อนไข i <= 5 อีกครั้ง ทำเช่นนี้ไปเรื่อยๆ จนกว่าผลการตรวจสอบเงื่อนไข i<=5 จะเป็นเท็จ หรือตัวแปร i จะมีค่าเท่ากับ 6 การทำงานของคำสั่ง for จะสิ้นสุดลง และข้ามไปทำคำสั่งบรรทัดที่ 9 (return)

บรรทัดที่ 7 พิมพ์ค่าของตัวแปร i ออกหน้าจอภาพ

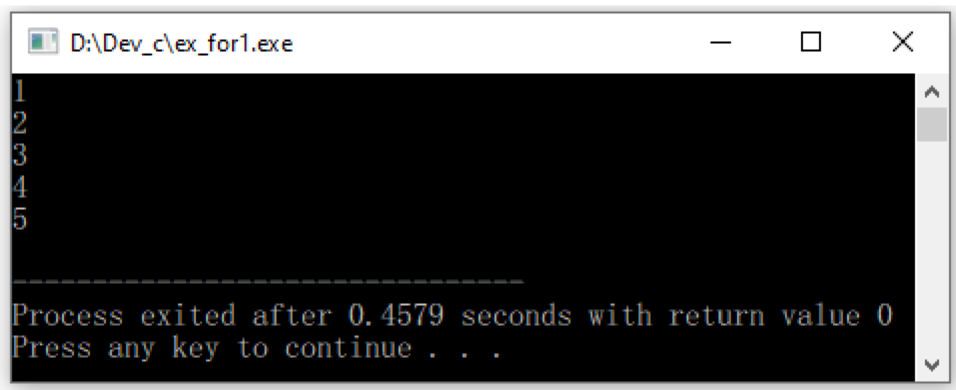
บรรทัดที่ 9 สิ้นสุดการทำงานของโปรแกรม การทำงานของโปรแกรมแสดงดังรูปที่ 8.2 ตารางที่ 8.1 และรูปที่ 8.3



รูปที่ 8.2 แผนผังแสดงการทำงานของโปรแกรมแสดงเลข 1 ถึง 5 ออกหน้าจอภาพ โดยคำสั่งวนรอบ for

ตารางที่ 8.1 การทำงานของโปรแกรมแสดงเลข 1 ถึง 5 ออกหน้าจอภาพ โดยคำสั่งวนรอบ for

ค่าตัวแปร i	เงื่อนไข i<=5	จำนวนครั้งการวนรอบ	หน้าจอภาพ
1	True	1	1
2	True	2	1 2
3	True	3	1 2 3
4	True	4	1 2 3 4
5	True	5	1 2 3 4 5
6	False	5	1 2 3 4 5



รูปที่ 8.3 ผลการทำงานของโปรแกรมแสดงเลข 1 ถึง 5 ออกหน้าจอภาพ โดยคำสั่งวนรอบ for

ตัวอย่าง โปรแกรมหาค่าผลรวมของเลข 1 ถึง 5 ออกหน้าจอภาพ โดยคำสั่งวนรอบ for

```
1  #include<stdio.h>
2  int main()
3  {
4      int sum, i;
5      sum = 0;
6      for(i = 1; i <= 5; i++)
7      {
8          sum = sum + i;
9      }
10     printf("Sum = %d\n",sum);
11     return(0);
12 }
```

จากตัวอย่างโปรแกรมสามารถอธิบายการทำงานของคำสั่งบางส่วนได้ดังนี้

บรรทัดที่ 4 ประกาศให้ sum และ i เป็นตัวแปรที่มีชนิดข้อมูลเป็นเลขจำนวนเต็ม

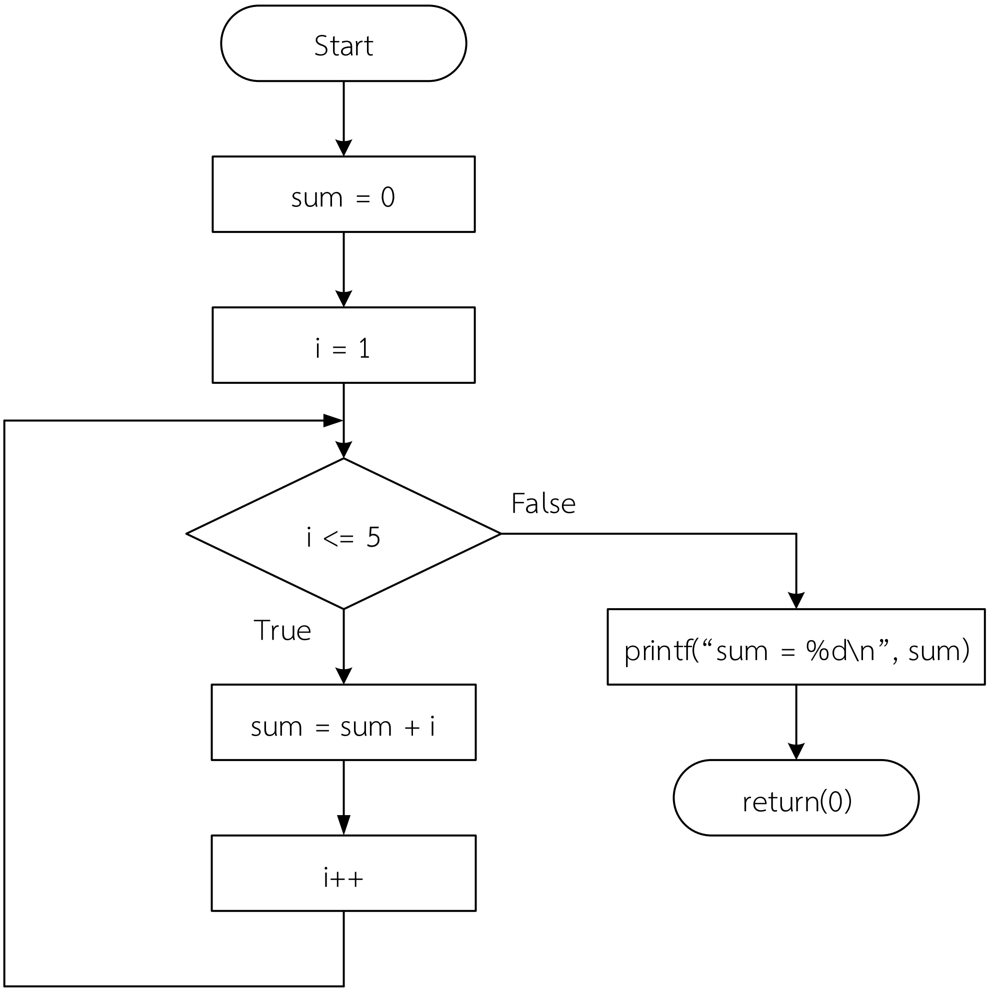
บรรทัดที่ 5 กำหนดให้ตัวแปร sum มีค่าเท่ากับ 0

บรรทัดที่ 6 คำสั่งวนรอบ for จะเริ่มต้นทำงานโดยกำหนดให้ตัวแปร i มีค่าเริ่มต้นเท่ากับ 1 (i = 1) จากนั้นทำการตรวจสอบเงื่อนไข i <= 5 ถ้าผลตรวจสอบเงื่อนไขเป็นจริง จะไปทำคำสั่งบรรทัดที่ 8 (sum = sum + 1) หลังจากทำคำสั่งบรรทัดที่ 8 เสร็จ จะทำการเพิ่มค่าตัวแปร i ขึ้นอีก 1 (i++) จากนั้นจะย้อนกลับไปทำการตรวจสอบเงื่อนไข i <= 5 อีกครั้ง ทำเช่นนี้ไปเรื่อยๆ จนกว่าผลการตรวจสอบเงื่อนไข i<=5 จะเป็นเท็จ หรือตัวแปร i จะมีค่าเท่ากับ 6 การทำงานของคำสั่ง for จะสิ้นสุดลง และข้ามไปทำคำสั่งบรรทัดที่ 10 (printf)

บรรทัดที่ 8 นำค่าของตัวแปร sum และ i มาบวกกัน และเก็บค่าผลบวกที่ได้ไว้ในตัวแปร sum

บรรทัดที่ 10 พิมพ์ค่าของตัวแปร sum ออกหน้าจอภาพ

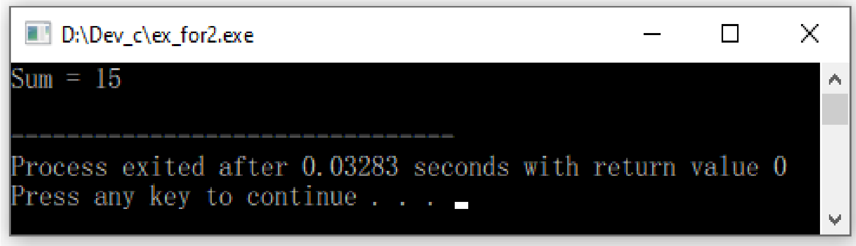
บรรทัดที่ 11 สิ้นสุดการทำงานของโปรแกรม การทำงานของโปรแกรมแสดงดังรูปที่ 8.4 ตารางที่ 8.2 และรูปที่ 8.5



รูปที่ 8.4 แผนผังแสดงการทำงานของโปรแกรมหาค่าผลรวมของเลข 1 ถึง 5 ออกหน้าจอภาพ โดยคำสั่งวนรอบ for

ตารางที่ 8.2 การทำงานของโปรแกรมหาค่าผลรวมของเลข 1 ถึง 5 ออกหน้าจอภาพ โดยคำสั่งวนรอบ for

ค่าตัวแปร i	เงื่อนไข i<=5	จำนวนครั้งการวนรอบ	ค่าตัวแปร sum	หน้าจอภาพ
1	-	0	0	
1	True	1	0 + 1 = 1	
2	True	2	0 + 1 + 2 = 3	
3	True	3	0 + 1 + 2 + 3 =6	
4	True	4	0 + 1 + 2 + 3 + 4 = 10	
5	True	5	0 + 1 + 2 + 3 + 4 + 5 = 15	
6	False	5		sum = 15



รูปที่ 8.5 ผลการทำงานของโปรแกรมหาค่าผลรวมของเลข 1 ถึง 5 ออกหน้าจอภาพ โดยคำสั่งวนรอบ for

8.3 คำสั่งวนรอบ while

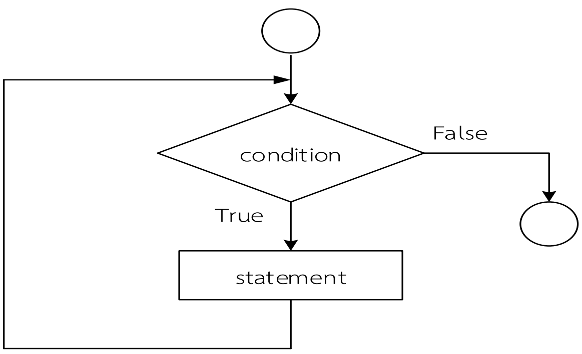
คำสั่งวนรอบ while เป็นคำสั่งที่ทำหน้าที่ควบคุมให้กลุ่มคำสั่งทำงานซ้ำไปจนกว่าผลการตรวจสอบเงื่อนไขที่กำหนดจะเป็นเท็จการทำงานจึงสิ้นสุด มีรูปแบบดังนี้

```
while(condition)
{
        statement;
}
```

โดยที่ condition คือ นิพจน์เงื่อนไขมีค่าจริงหรือเท็จอย่างใดอย่างหนึ่งเท่านั้น

statement คือ กลุ่มคำสั่งที่กำหนดให้ทำงานซ้ำ เมื่อผลการตรวจสอบเงื่อนไขเป็นจริง

การทำงานของคำสั่งวนรอบ while จากแผนผังในรูปที่ 8.6 เมื่อคำสั่ง while เริ่มทำงาน เงื่อนไขจะถูกตรวจสอบ ถ้าผลการตรวจสอบเป็นจริงกลุ่มคำสั่งที่กำหนดให้ทำงานซ้ำจะทำงาน จากนั้นจะย้อนกลับไปตรวจสอบเงื่อนไขอีกครั้ง ถ้าผลการตรวจสอบเงื่อนไขเป็นจริงกลุ่มคำสั่งที่กำหนดให้ทำงานซ้ำจะทำงานอีกครั้ง แต่ถ้าผลการตรวจสอบเป็นเท็จการทำงานของคำสั่ง while จะสิ้นสุดลง และข้ามไปทำคำสั่งที่อยู่ถัดไป



รูปที่ 8.6 แผนผังแสดงการทำงานของคำสั่งวนรอบ while

ตัวอย่างโปรแกรมแสดงเลข 1 ถึง 5 ออกหน้าจอภาพ โดยคำสั่งวนรอบ while

```
1  #include<stdio.h>
2  int main()
3  {
4      int i;
5      i = 1;
6      while(i <= 5)
7      {
8          printf("%d\n",i);
9          i = i + 1;
10     }
11     return(0);
12 }
```

จากตัวอย่างโปรแกรมสามารถอธิบายการทำงานของคำสั่งบางส่วนได้ดังนี้

บรรทัดที่ 4 ประกาศให้ i เป็นตัวแปรที่มีชนิดข้อมูลเป็นเลขจำนวนเต็ม

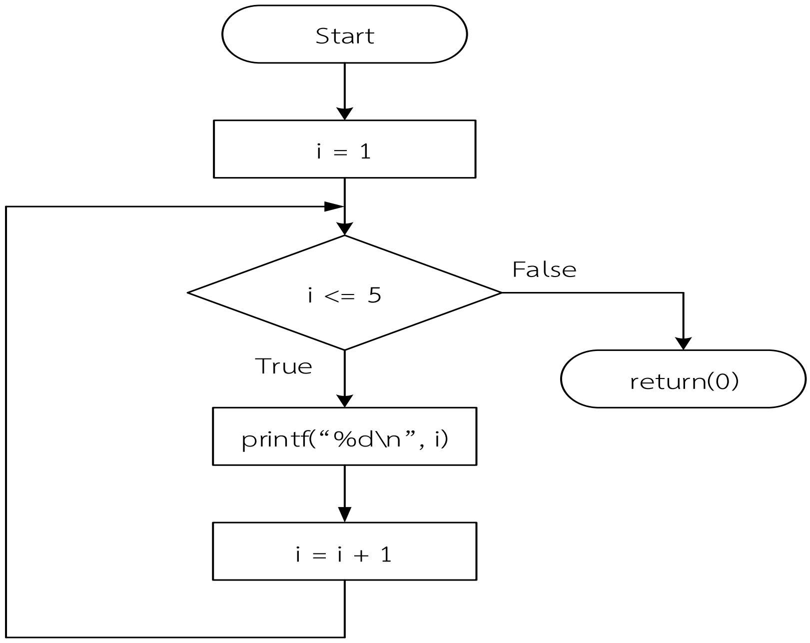
บรรทัดที่ 5 กำหนดให้ตัวแปร i มีค่าเริ่มต้นเท่ากับ 1 (i = 1)

บรรทัดที่ 6 คำสั่งวนรอบ while จะเริ่มต้นทำงาน โดยทำการตรวจสอบเงื่อนไข i <= 5 ถ้าผลตรวจสอบเงื่อนไขเป็นจริง จะไปทำคำสั่งบรรทัดที่ 8 (printf) และคำสั่งบรรทัดที่ 9 (i = i + 1) หลังจากทำคำสั่งบรรทัดที่ 9 เสร็จ จะย้อนกลับไปทำการตรวจสอบเงื่อนไข i <= 5 อีกครั้ง ทำเช่นนี้ไปเรื่อยๆ จนกว่าผลการตรวจสอบเงื่อนไข i<=5 จะเป็นเท็จ หรือตัวแปร i จะมีค่าเท่ากับ 6 การทำงานของคำสั่ง while จึงจะสิ้นสุดลง และข้ามไปทำคำสั่งบรรทัดที่ 11 (return)

บรรทัดที่ 8 พิมพ์ค่าของตัวแปร i ออกหน้าจอภาพ

บรรทัดที่ 9 เพิ่มค่าของตัวแปร i ขึ้นเท่ากับ 1

บรรทัดที่ 11 สิ้นสุดการทำงานของโปรแกรม การทำงานของโปรแกรมแสดงดังรูปที่ 8.6 ตารางที่ 8.3 และรูปที่ 8.7

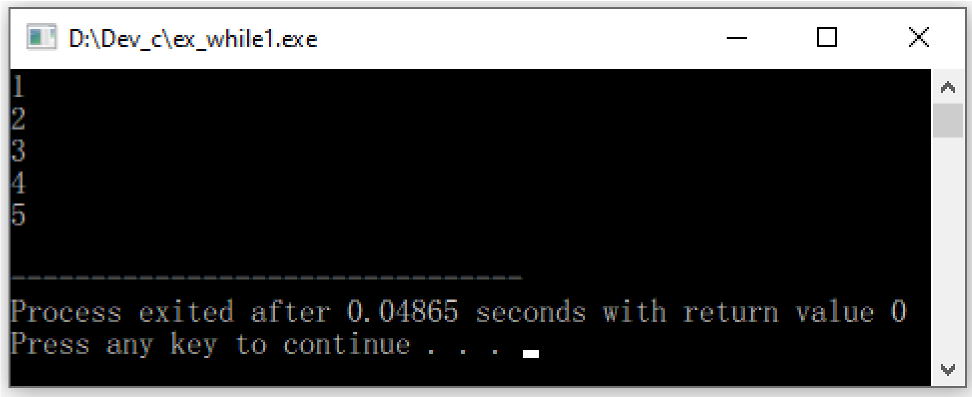


รูปที่ 8.6 แผนผังแสดงการทำงานของโปรแกรมแสดงเลข 1 ถึง 5 ออกหน้าจอภาพ โดยคำสั่งวนรอบ while

ตารางที่ 8.3 การทำงานของโปรแกรมแสดงเลข 1 ถึง 5 ออกหน้าจอภาพ โดยคำสั่งวนรอบ while

ค่าตัวแปร i	เงื่อนไข i<=5	จำนวนครั้งการวนรอบ	หน้าจอภาพ
1	True	1	1
2	True	2	1 2
3	True	3	1 2 3
4	True	4	1 2 3 4
5	True	5	1 2 3 4 5

6	False	5	1 2 3 4 5



รูปที่ 8.7 ผลการทำงานของโปรแกรมแสดงเลข 1 ถึง 5 ออกหน้าจอภาพ โดยคำสั่งวนรอบ while

ตัวอย่างโปรแกรมหาค่าผลรวมของเลข 1 ถึง 5 ออกหน้าจอภาพ โดยคำสั่งวนรอบ while

```
1  #include<stdio.h>
2  int main()
3  {
4      int sum, i;
5      sum = 0;
6      i = 1;
7      while(i <= 5)
8      {
9          sum = sum + i;
10         i = i + 1;
11     }
12     printf("Sum = %d\n",sum);
13     return(0);
14 }
```

จากตัวอย่างโปรแกรมสามารถอธิบายการทำงานของคำสั่งบางส่วนได้ดังนี้

บรรทัดที่ 4 ประกาศให้ sum และ i เป็นตัวแปรที่มีชนิดข้อมูลเป็นเลขจำนวนเต็ม

บรรทัดที่ 5 กำหนดให้ตัวแปร sum มีค่าเท่ากับ 0

บรรทัดที่ 6 กำหนดให้ตัวแปร i มีค่าเท่ากับ 1

บรรทัดที่ 7 คำสั่งวนรอบ while จะเริ่มต้นทำงานโดยทำการตรวจสอบเงื่อนไข i <= 5 ถ้าผลตรวจสอบเงื่อนไขเป็นจริง จะไปทำคำสั่งบรรทัดที่ 9 (sum = sum + i) และคำสั่งบรรทัดที่ 10 (i = i + 1)

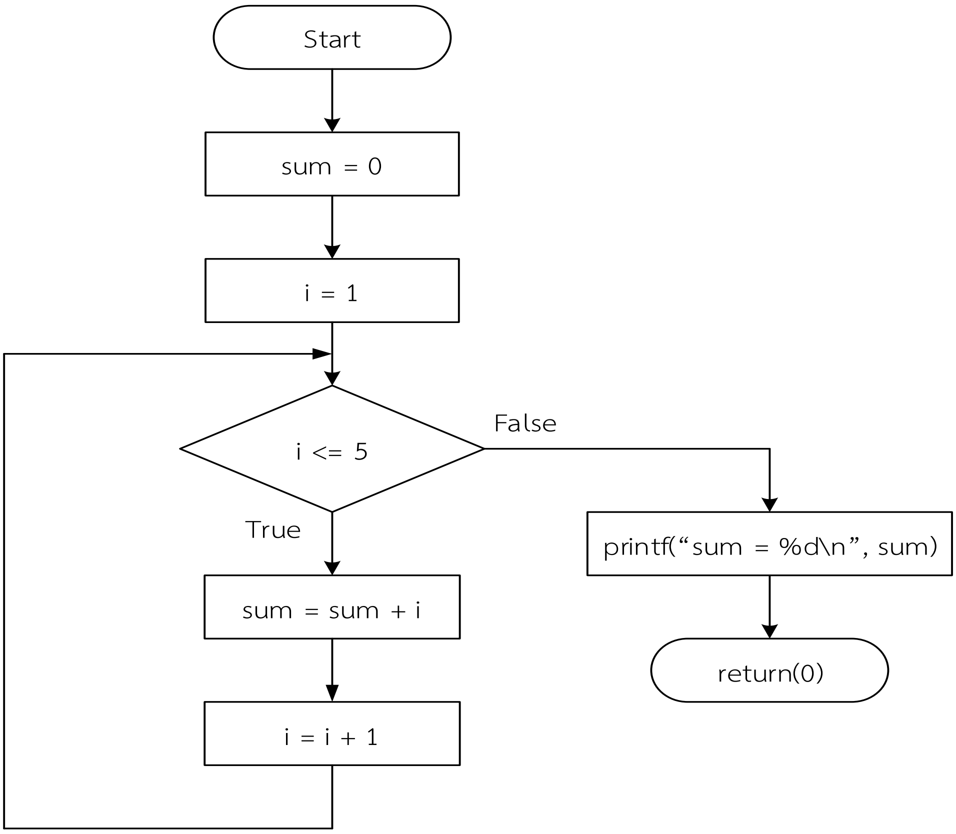
หลังจากทำคำสั่งบรรทัดที่ 10 เสร็จ จะย้อนกลับไปทำการตรวจสอบเงื่อนไข i <= 5 อีกครั้ง ทำเช่นนี้ไปเรื่อยๆ จนกว่าผลการตรวจสอบเงื่อนไข i<=5 จะเป็นเท็จ หรือตัวแปร i จะมีค่าเท่ากับ 6 การทำงานของคำสั่ง while จึงจะสิ้นสุดลง และข้ามไปทำคำสั่งบรรทัดที่ 12 (printf)

บรรทัดที่ 9 นำค่าของตัวแปร sum และ i มาบวกกัน และเก็บค่าผลบวกที่ได้ไว้ในตัวแปร sum

บรรทัดที่ 10 เพิ่มค่าของตัวแปร i ขึ้นเท่ากับ 1

บรรทัดที่ 12 พิมพ์ค่าของตัวแปร sum ออกหน้าจอภาพ

บรรทัดที่ 13 สิ้นสุดการทำงานของโปรแกรม การทำงานของโปรแกรมแสดงดังรูปที่ 8.8 ตารางที่ 8.4 และรูปที่ 8.9

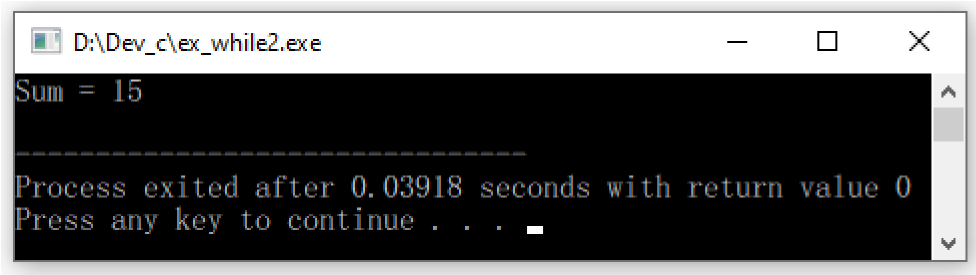


รูปที่ 8.8 แผนผังแสดงการทำงานของโปรแกรมหาค่าผลรวมของเลข 1 ถึง 5 ออกหน้าจอภาพ โดยคำสั่งวนรอบ while

ตารางที่ 8.4 การทำงานของโปรแกรมหาค่าผลรวมของเลข 1 ถึง 5 ออกหน้าจอภาพ โดยคำสั่งวนรอบ while

ค่าตัวแปร i	เงื่อนไข i<=5	จำนวนครั้งการวนรอบ	ค่าตัวแปร sum	หน้าจอภาพ
1	-	0	0	
1	True	1	0 + 1 = 1	

2	True	2	0 + 1 + 2 = 3	
3	True	3	0 + 1 + 2 + 3 =6	
4	True	4	0 + 1 + 2 + 3 + 4 = 10	
5	True	5	0 + 1 + 2 + 3 + 4 + 5 = 15	
6	False	5		sum = 15



รูปที่ 8.9 ผลการทำงานของโปรแกรมหาค่าผลรวมของเลข 1 ถึง 5 ออกหน้าจอภาพ โดยคำสั่งวนรอบ while

8.4 คำสั่งวนรอบ do...while

คำสั่งวนรอบ **do...**while เป็นคำสั่งที่ทำหน้าที่ควบคุมให้กลุ่มคำสั่งทำงานซ้ำไปจนกว่าผลการตรวจสอบเงื่อนไขที่กำหนดจะเป็นเท็จการทำงานจึงสิ้นสุด เหมือนกับคำสั่งวนรอบ while แต่จะแตกต่างตรงคำสั่งวนรอบ do...while จะเข้าไปทำคำสั่งที่กำหนดให้ทำงานซ้ำก่อน จากนั้นจึงจะทำการตรวจสอบเงื่อนไข มีรูปแบบคำสั่งดังนี้

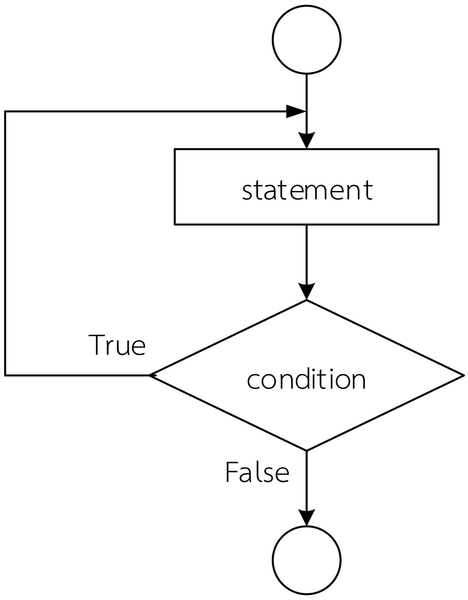
```
do
{
        statement;
} while(condition);
```

โดยที่ condition คือ นิพจน์เงื่อนไขมีค่าจริงหรือเท็จอย่างใดอย่างหนึ่งเท่านั้น

statement คือ กลุ่มคำสั่งที่กำหนดให้ทำงานซ้ำ เมื่อผลการตรวจสอบเงื่อนไขเป็นจริง

การทำงานของคำสั่งวนรอบ do...while จากแผนผังในรูปที่ 8.10 เมื่อคำสั่ง do...while เริ่มทำงาน กลุ่มคำสั่งที่กำหนดให้ทำงานซ้ำจะทำงาน จากนั้นเงื่อนไขจะถูกตรวจสอบ ถ้าผลการตรวจสอบเป็นจริงกลุ่ม

คำสั่งที่กำหนดให้ทำงานซ้ำจะทำงานซ้ำอีกหนึ่งครั้งและย้อนกลับไปตรวจสอบเงื่อนไขอีกครั้ง แต่ถ้าผลการตรวจสอบเป็นเท็จการทำงานของคำสั่ง do...while จะสิ้นสุดลง และข้ามไปทำคำสั่งที่อยู่ถัดไป



รูปที่ 8.10 แผนผังแสดงการทำงานของคำสั่งวนรอบ do...while

ตัวอย่างโปรแกรมแสดงเลข 1 ถึง 5 ออกหน้าจอภาพ โดยคำสั่งวนรอบ do...while

```
1  #include<stdio.h>
2  int main()
3  {
4      int i;
5      i = 1;
6      do
7      {
8          printf("%d\n",i);
9          i = i + 1;
10     } while(i <= 5);
11     return(0);
12 }
```

จากตัวอย่างโปรแกรมสามารถอธิบายการทำงานของคำสั่งบางส่วนได้ดังนี้

บรรทัดที่ 4 ประกาศให้ i เป็นตัวแปรที่มีชนิดข้อมูลเป็นเลขจำนวนเต็ม

บรรทัดที่ 5 กำหนดให้ตัวแปร i มีค่าเริ่มต้นเท่ากับ 1 (i = 1)

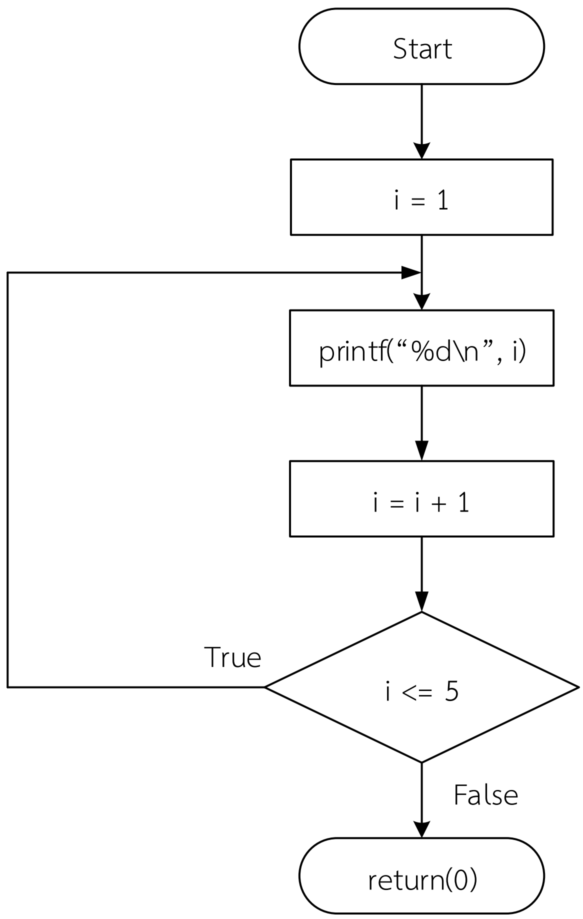
บรรทัดที่ 6 คำสั่งวนรอบ do...while เริ่มต้นทำงาน โดยไปคำสั่งบรรทัดที่ 8 (printf) และคำสั่งบรรทัดที่ 9 (i = i + 1)

บรรทัดที่ 8 พิมพ์ค่าของตัวแปร i ออกหน้าจอภาพ

บรรทัดที่ 9 เพิ่มค่าของตัวแปร i ขึ้นเท่ากับ 1

บรรทัดที่ 10 ทำการตรวจสอบเงื่อนไข i <= 5 ที่อยู่หลัง while ถ้าผลตรวจสอบเงื่อนไขเป็นจริง จะย้อนกลับไปทำคำสั่งบรรทัดที่ 8 (printf) และคำสั่งบรรทัดที่ 9 (i = i + 1) อีกหนึ่งครั้ง หลังจากทำคำสั่งบรรทัดที่ 9 เสร็จ จะทำการตรวจสอบเงื่อนไข i <= 5 ที่อยู่หลัง while อีกครั้ง ทำเช่นนี้ไปเรื่อยๆ จนกว่าผลการตรวจสอบเงื่อนไข i<=5 จะเป็นเท็จ หรือตัวแปร i จะมีค่าเท่ากับ 6 การทำงานของคำสั่ง do…while จึงจะสิ้นสุดลง และข้ามไปทำคำสั่งบรรทัดที่ 11 (return)

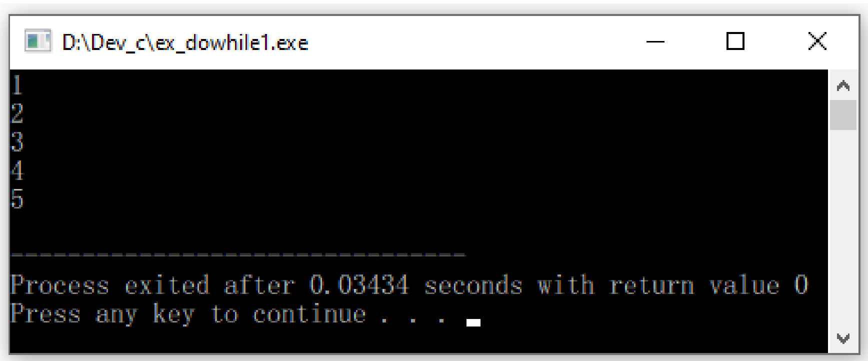
บรรทัดที่ 11 สิ้นสุดการทำงานของโปรแกรม การทำงานของโปรแกรมแสดงดังรูปที่ 8.11 ตารางที่ 8.5 และรูปที่ 8.12



รูปที่ 8.11 แผนผังแสดงการทำงานของโปรแกรมแสดงเลข 1 ถึง 5 ออกหน้าจอภาพ โดยคำสั่งวนรอบ do…while

ตารางที่ 8.5 การทำงานของโปรแกรมแสดงเลข 1 ถึง 5 ออกหน้าจอภาพ โดยคำสั่งวนรอบ do...while

ค่าตัวแปร i	เงื่อนไข i<=5	จำนวนครั้งการวนรอบ	หน้าจอภาพ
1		1	1
2	True	2	1 2
3	True	3	1 2 3
4	True	4	1 2 3 4
5	True	5	1 2 3 4 5
6	False	5	1 2 3 4 5



รูปที่ 8.12 ผลการทำงานของโปรแกรมแสดงเลข 1 ถึง 5 ออกหน้าจอภาพ โดยคำสั่งวนรอบ do...while

ตัวอย่างโปรแกรมหาค่าผลรวมของเลข 1 ถึง 5 ออกหน้าจอภาพ โดยคำสั่งวนรอบ do...while

```
1  #include<stdio.h>
2  int main()
3  {
4      int sum, i;
5      sum = 0;
6      i = 1;
7      do
8      {
9          sum = sum + i;
10         i = i + 1;
11     }while(i <= 5);
12     printf("Sum = %d\n",sum);
13     return(0);
14 }
```

จากตัวอย่างโปรแกรมสามารถอธิบายการทำงานของคำสั่งบางส่วนได้ดังนี้

บรรทัดที่ 4 ประกาศให้ sum และ i เป็นตัวแปรที่มีชนิดข้อมูลเป็นเลขจำนวนเต็ม

บรรทัดที่ 5 กำหนดให้ตัวแปร sum มีค่าเท่ากับ 0

บรรทัดที่ 6 กำหนดให้ตัวแปร i มีค่าเท่ากับ 1

บรรทัดที่ 6 คำสั่งวนรอบ do...while เริ่มต้นทำงาน โดยไปคำสั่งบรรทัดที่ 9 (sum = sum + i) และคำสั่งบรรทัดที่ 10 (i = i + 1)

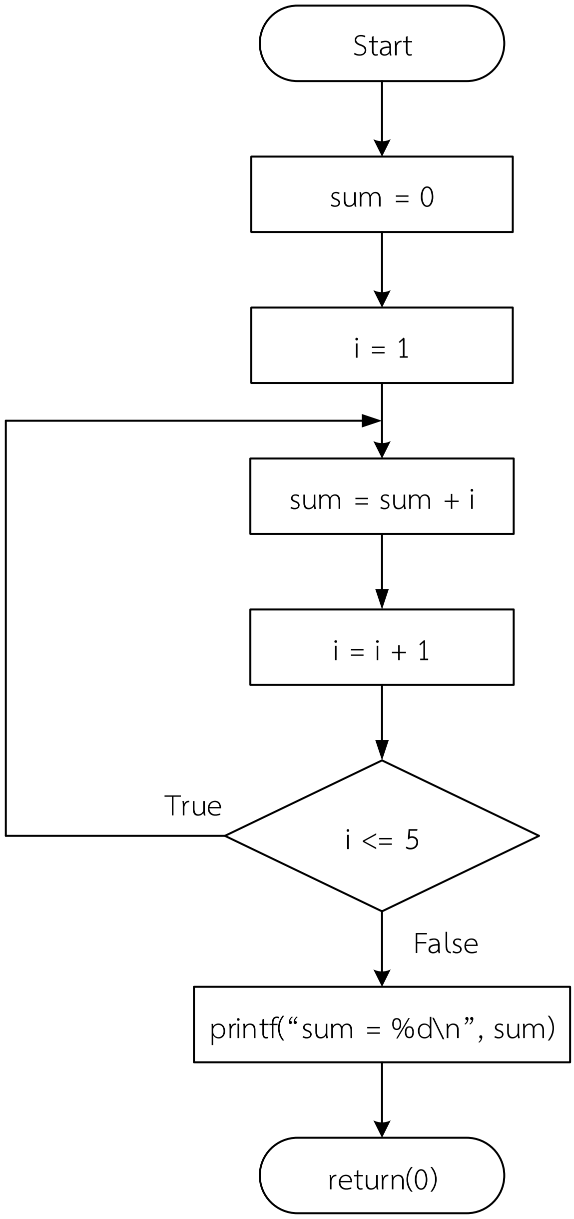
บรรทัดที่ 9 นำค่าของตัวแปร sum และ i มาบวกกัน และเก็บค่าผลบวกที่ได้ไว้ในตัวแปร sum

บรรทัดที่ 10 เพิ่มค่าของตัวแปร i ขึ้นเท่ากับ 1

บรรทัดที่ 11 ทำการตรวจสอบเงื่อนไข i <= 5 ที่อยู่หลัง while ถ้าผลตรวจสอบเงื่อนไขเป็นจริง จะย้อนกลับไปทำคำสั่งบรรทัดที่ 9 (sum = sum + i) และคำสั่งบรรทัดที่ 10 (i = i + 1) อีกหนึ่งครั้ง หลังจากทำคำสั่งบรรทัดที่ 10 เสร็จ จะทำการตรวจสอบเงื่อนไข i <= 5 ที่อยู่หลัง while อีกครั้ง ทำเช่นนี้ไปเรื่อยๆ จนกว่าผลการตรวจสอบเงื่อนไข i<=5 จะเป็นเท็จ หรือตัวแปร i จะมีค่าเท่ากับ 6 การทำงานของคำสั่ง do...while จึงจะสิ้นสุดลง และข้ามไปทำคำสั่งบรรทัดที่ 12 (printf)

บรรทัดที่ 12 พิมพ์ค่าของตัวแปร sum ออกหน้าจอภาพ

บรรทัดที่ 13 สิ้นสุดการทำงานของโปรแกรม การทำงานของโปรแกรมแสดงดังรูปที่ 8.13 ตารางที่ 8.6 และรูปที่ 8.14

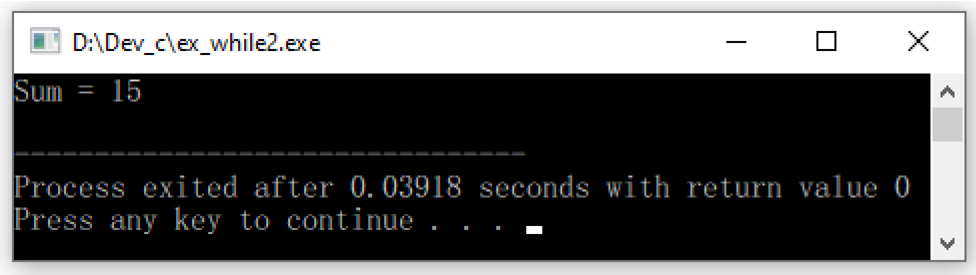


รูปที่ 8.13 แผนผังแสดงการทำงานของโปรแกรมหาค่าผลรวมของเลข 1 ถึง 5 ออกหน้าจอภาพ โดยคำสั่งวนรอบ do…while

ตารางที่ 8.6 การทำงานของโปรแกรมหาค่าผลรวมของเลข 1 ถึง 5 ออกหน้าจอภาพ โดยคำสั่งวนรอบ do…while

ค่าตัวแปร i	เงื่อนไข i<=5	จำนวนครั้งการวนรอบ	ค่าตัวแปร sum	หน้าจอภาพ
1	-	0	0	
1	-	1	0 + 1 = 1	

2	True	2	0 + 1 + 2 = 3	
3	True	3	0 + 1 + 2 + 3 =6	
4	True	4	0 + 1 + 2 + 3 + 4 = 10	
5	True	5	0 + 1 + 2 + 3 + 4 + 5 = 15	
6	False	5		sum = 15



รูปที่ 8.14 ผลการทำงานของโปรแกรมหาค่าผลรวมของเลข 1 ถึง 5 ออกหน้าจอภาพ โดยคำสั่งวนรอบ do...while

8.5 คำสั่งวนรอบแบบซ้อน

คำสั่งวนรอบแบบซ้อน เป็นคำสั่งที่ทำหน้าที่ควบคุมให้คำสั่งทำงานซ้ำทำงานที่มีรูปแบบซับซ้อน โดยนำคำสั่งวนรอบ for while และ do...while หลายคำสั่งมาเขียนซ้อนกัน เหมือนกับการเขียนคำสั่งเลือกทำ if แบบซ้อน มีรูปแบบคำสั่งดังนี้

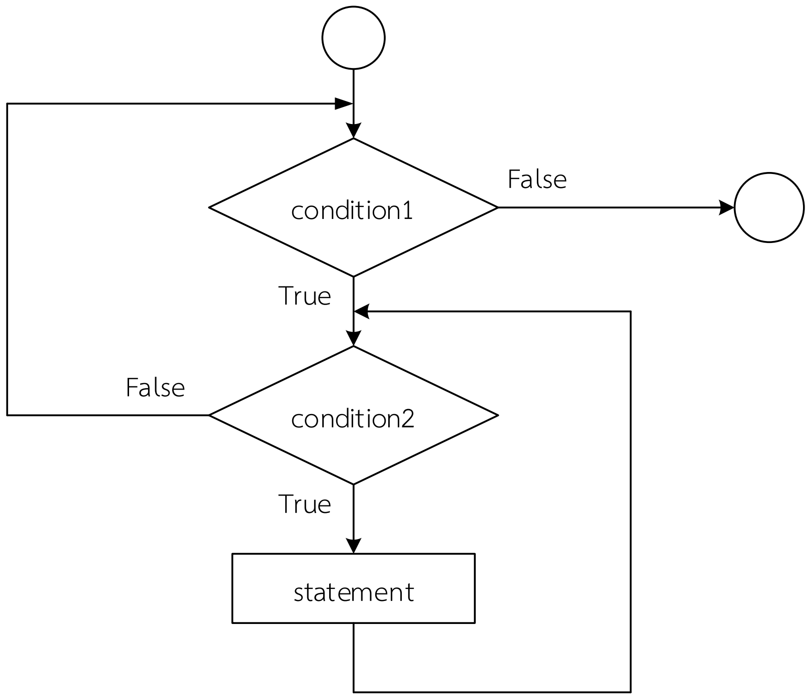
```
Loop_statement1 (condition1)
{
 Loop_statement2(condition2)
         {
                  statement;
         }
}
```

โดยที่ Loop_statement1, Loop_statement2 คือ คำสั่งวนรอบ for while หรือ do...while

condition1, condition2 คือ นิพจน์เงื่อนไขมีค่าจริงหรือเท็จอย่างใดอย่างหนึ่งเท่านั้น

statement คือ กลุ่มคำสั่งที่กำหนดให้ทำงานซ้ำ เมื่อผลการตรวจสอบเงื่อนไขเป็นจริง

การทำงานของคำสั่งวนรอบแบบซ้อน จากแผนผังในรูปที่ 8.15 การทำงานจะเริ่มจากคำสั่งวนรอบที่อยู่ข้างนอก เงื่อนไข condition1 จะถูกตรวจสอบ ถ้าผลการตรวจสอบเป็นจริง คำสั่งวนรอบที่อยู่ข้างในจะทำงาน จากนั้นเงื่อนไข condition2 จะถูกตรวจสอบ ถ้าผลการตรวจสอบเป็นจริง กลุ่มคำสั่งที่กำหนดให้ทำงานซ้ำจะทำงาน จากนั้นจะย้อนกลับไปตรวจสอบเงื่อนไข condition2 อีกครั้ง แต่ถ้าผลการตรวจสอบเป็นเท็จการทำงานของคำสั่งวนรอบที่อยู่ข้างในจะสิ้นสุดลง จากนั้นจะย้อนกลับไปทำคำสั่งวนรอบที่อยู่ข้างนอกอีกครั้ง และทำงานอย่างนี้ไปเรื่อยๆ จนกว่าผลการตรวจสอบเงื่อนไข condition1 จะเป็นเท็จการทำงานของคำสั่งวนรอบแบบซ้อนจึงจะสิ้นสุดลง และข้ามไปทำคำสั่งที่อยู่ถัดไป



รูปที่ 8.15 แผนผังแสดงการทำงานของคำสั่งวนรอบแบบซ้อน

ตัวอย่างโปรแกรมแสดงตัวเลขออกหน้าจอภาพจำนวน 5 บรรทัด โดยคำสั่งวนรอบ for ซ้อนกัน ซึ่งโปรแกรมมีผลการทำงานดังนี้

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

```
1  #include<stdio.h>
2  int main()
3  {
4      int i, j;
5      for(i=1; i <= 5; i++)
6      {
7          for(j=1; j <= i; j++)
8          {
9              printf("%d ",j);
10         }
11         printf("\n");
12     }
13     return(0);
14 }
```

จากตัวอย่างโปรแกรมสามารถอธิบายการทำงานของคำสั่งบางส่วนได้ดังนี้

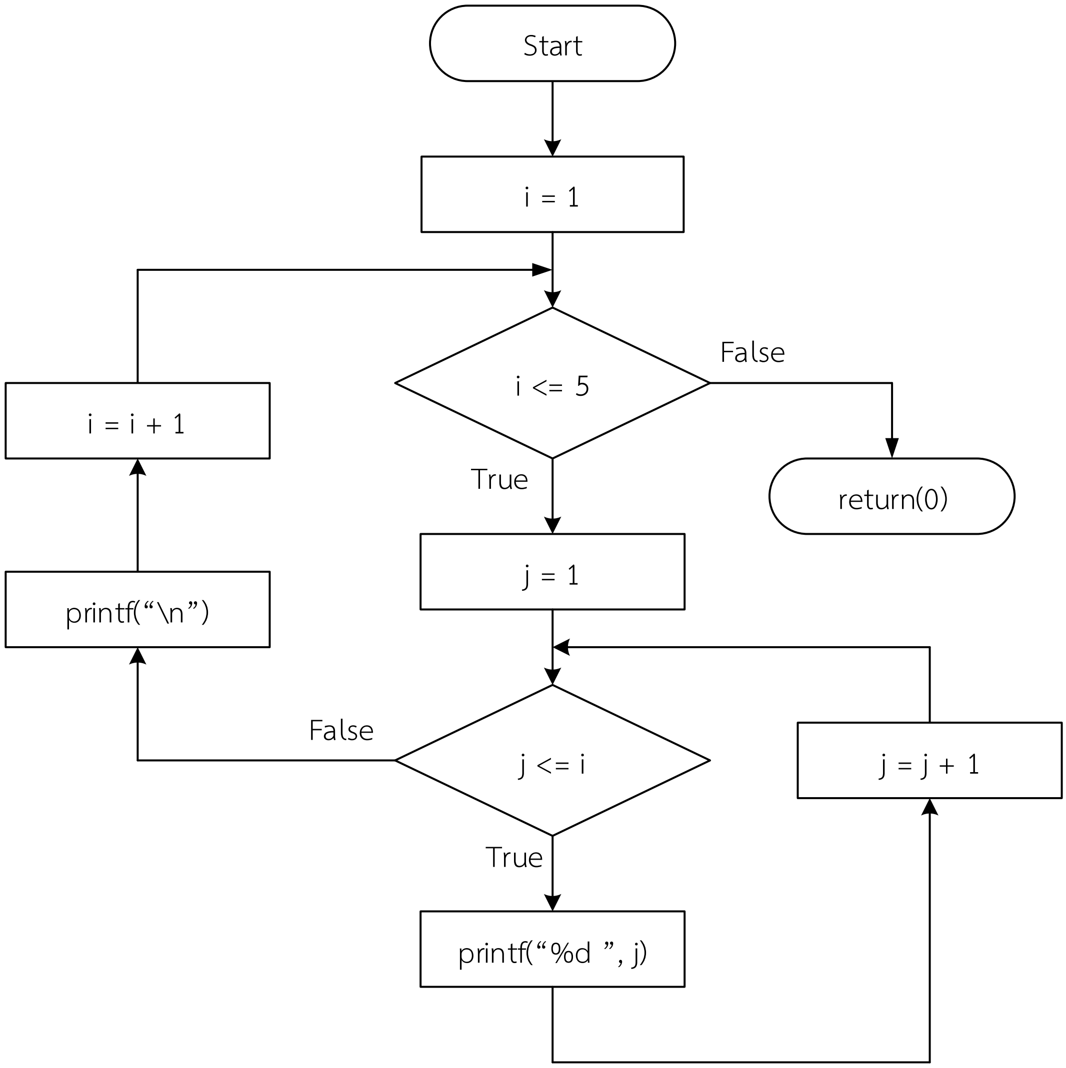
บรรทัดที่ 4 ประกาศให้ i และ j เป็นตัวแปรที่มีชนิดข้อมูลเป็นเลขจำนวนเต็ม

บรรทัดที่ 5 คำสั่งวนรอบ for ที่อยู่ข้างนอก จะเริ่มต้นทำงานโดยกำหนดให้ตัวแปร i มีค่าเริ่มต้นเท่ากับ 1 (i = 1) จากนั้นทำการตรวจสอบเงื่อนไข i <= 5 ถ้าผลตรวจสอบเงื่อนไขเป็นจริง จะไปทำคำสั่งบรรทัดที่ 7 (for) ซึ่งเป็นคำสั่งวนรอบที่อยู่ข้างใน หลังจากทำคำสั่งบรรทัดที่ 7 เสร็จ จะไปทำคำสั่งบรรทัดที่ 11 ซึ่งเป็นคำสั่งทำการเลื่อนไปยังบรรทัดถัดไป จากนั้นจะทำการเพิ่มค่าตัวแปร i ขึ้นอีก 1 (i++) จากนั้นจะย้อนกลับไปทำการตรวจสอบเงื่อนไข i <= 5 อีกครั้ง ทำเช่นนี้ไปเรื่อยๆ จนกว่าผลการตรวจสอบเงื่อนไข i<=5 จะเป็นเท็จ หรือตัวแปร i จะมีค่าเท่ากับ 6 การทำงานของคำสั่ง for ที่อยู่ข้างนอกจะสิ้นสุดลง และข้ามไปทำคำสั่งบรรทัดที่ 13 (return)

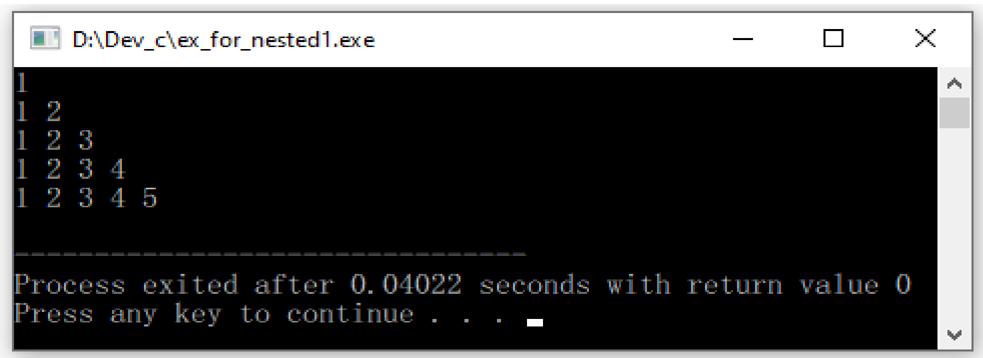
บรรทัดที่ 7 คำสั่งวนรอบ for ที่อยู่ข้างใน จะเริ่มต้นทำงานโดยกำหนดให้ตัวแปร j มีค่าเริ่มต้นเท่ากับ 1 (j = 1) จากนั้นทำการตรวจสอบเงื่อนไข j <= i ถ้าผลตรวจสอบเงื่อนไขเป็นจริง จะไปทำคำสั่งบรรทัดที่ 9 (printf) หลังจากทำคำสั่งบรรทัดที่ 9 เสร็จ จะทำการเพิ่มค่าตัวแปร i ขึ้นอีก 1 (i++) จากนั้นจะย้อนกลับไปทำการตรวจสอบเงื่อนไข j <= i อีกครั้ง ทำเช่นนี้ไปเรื่อยๆ จนกว่าผลการตรวจสอบเงื่อนไข j <= i จะเป็นเท็จ หรือตัวแปร j จะมีค่ามากกว่าค่าของตัวแปร i การทำงานของคำสั่ง for ที่อยู่ข้างในจึงจะสิ้นสุดลง และข้ามไปทำคำสั่งบรรทัดที่ 11 (printf)

บรรทัดที่ 9 พิมพ์ค่าของตัวแปร i ออกหน้าจอภาพ

บรรทัดที่ 13 สิ้นสุดการทำงานของโปรแกรม การทำงานของโปรแกรมแสดงดังรูปที่ 8.16 และรูปที่ 8.17



รูปที่ 8.16 แผนผังแสดงการทำงานของโปรแกรมแสดงตัวเลขออกหน้าจอภาพจำนวน 5 บรรทัด โดยคำสั่งวนรอบ for ซ้อนกัน



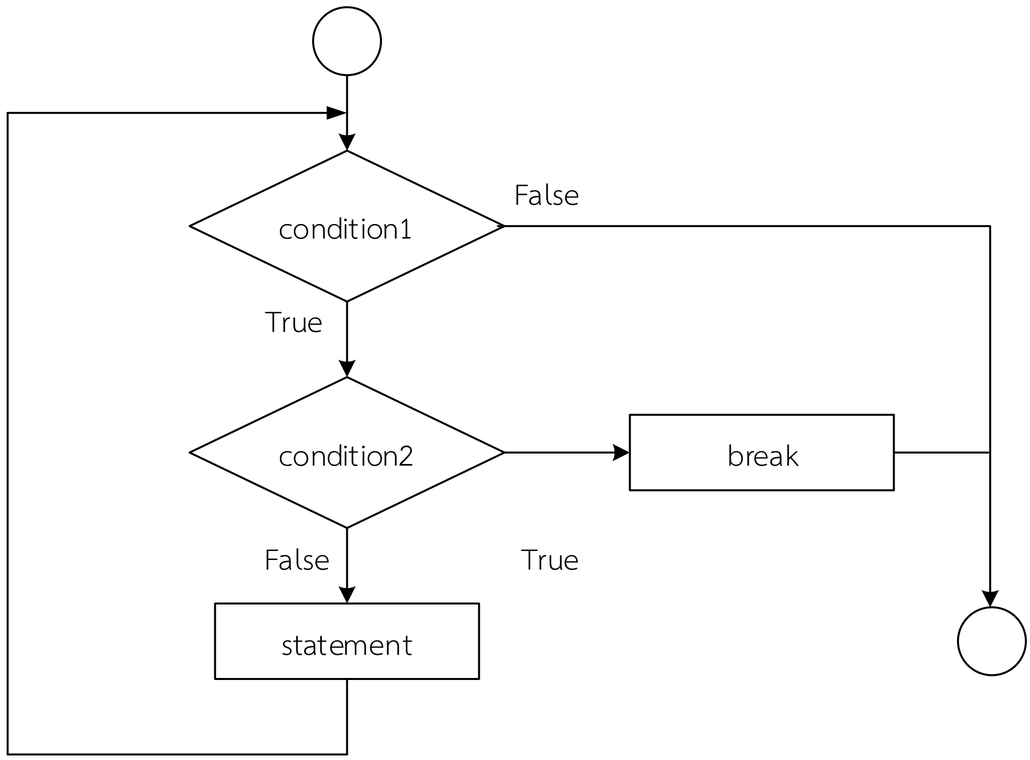
รูปที่ 8.17 ผลการทำงานของโปรแกรมแสดงตัวเลขออกหน้าจอภาพจำนวน 5 บรรทัด โดยคำสั่งวนรอบ for ซ้อนกัน

8.6 คำสั่ง break

คำสั่ง break เป็นคำสั่งที่ทำหน้าที่ควบคุมให้คำสั่งวนรอบหยุดการทำงานทันทีและย้ายการทำงานไปยังคำสั่งที่อยู่ถัดจากคำสั่งวนรอบ นิยมใช้หยุดการทำงานของคำสั่งวนรอบ เมื่อผลการตรวจสอบเงื่อนไขที่ต้องการเป็นจริง กรณีมีรูปแบบคำสั่งดังนี้

```
break;
```

การทำงานของคำสั่ง break จากแผนผังในรูปที่ 8.18 เมื่อผลการตรวจสอบเงื่อนไข condition2 เป็นจริง คำสั่ง break จะทำงาน จากนั้นคำสั่ง break จะทำให้การทำงานของคำสั่งวนรอบหยุดลง และข้ามไปทำคำสั่งที่อยู่ถัดไป แม้ว่าผลการตรวจสอบเงื่อนไข condition1 จะไม่เป็นเท็จก็ตาม



รูปที่ 8.18 แผนผังแสดงการทำงานของคำสั่ง break

ตัวอย่างโปรแกรมแสดงเลข 1 ถึง 3 ออกหน้าจอภาพ โดยใช้คำสั่งวนรอบ while และคำสั่ง break

```
1  #include<stdio.h>
2  int main()
3  {
4      int i;
5      i = 1;
6      while(i <= 5)
7      {
8          printf("%d\n",i);
9          i = i + 1;
10         if(i > 3)
11         {
12             break;
13         }
14     }
15     return(0);
16 }
```

จากตัวอย่างโปรแกรมสามารถอธิบายการทำงานของคำสั่งบางส่วนได้ดังนี้

บรรทัดที่ 4 ประกาศให้ i เป็นตัวแปรที่มีชนิดข้อมูลเป็นเลขจำนวนเต็ม

บรรทัดที่ 5 กำหนดให้ตัวแปร i มีค่าเริ่มต้นเท่ากับ 1 (i = 1)

บรรทัดที่ 6 คำสั่งวนรอบ while เริ่มต้นทำงาน โดยทำการตรวจสอบเงื่อนไข i <= 5 ถ้าผลตรวจสอบเงื่อนไขเป็นจริง จะไปทำคำสั่งบรรทัดที่ 8 - 12 หลังจากทำคำสั่งเสร็จ จะย้อนกลับไปทำการตรวจสอบเงื่อนไข i <= 5 อีกครั้ง ทำเช่นนี้ไปเรื่อยๆ จนกว่าผลการตรวจสอบเงื่อนไข i<=5 จะเป็นเท็จ หรือตัวแปร i จะมีค่าเท่ากับ 6 การทำงานของคำสั่ง while จึงจะสิ้นสุดลง และข้ามไปทำคำสั่งบรรทัดที่ 15 (return)

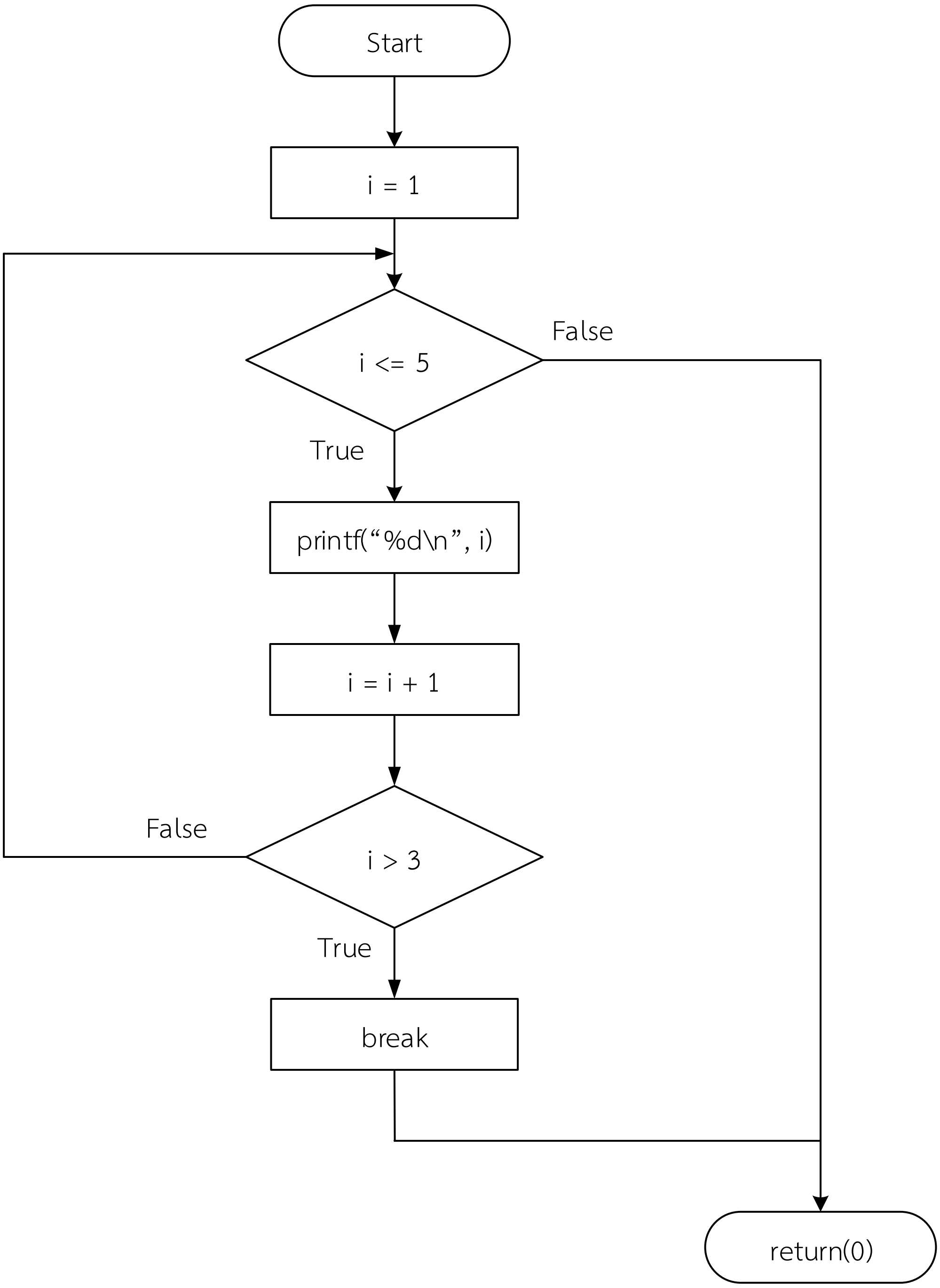
บรรทัดที่ 8 พิมพ์ค่าของตัวแปร i ออกหน้าจอภาพ

บรรทัดที่ 9 เพิ่มค่าของตัวแปร i ขึ้นเท่ากับ 1

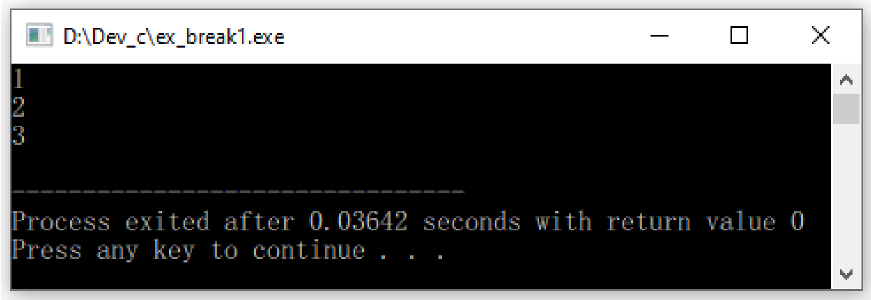
บรรทัดที่ 10 คำสั่ง if ทำการตรวจเงื่อนไข i > 3 ถ้าผลตรวจสอบเงื่อนไขเป็นจริง จะไปทำคำสั่งบรรทัดที่ 12 (break)

บรรทัดที่ 12 คำสั่ง break ให้คำสั่งวนรอบ while หยุดการทำงานทันทีและย้ายการทำงานไปยังคำสั่งบรรทัดที่ 15 (return)

บรรทัดที่ 15 สิ้นสุดการทำงานของโปรแกรม การทำงานของโปรแกรมแสดงดังรูปที่ 8.19 และรูปที่ 8.20



รูปที่ 8.19 แผนผังแสดงการทำงานของโปรแกรมแสดงเลข 1 ถึง 3 ออกหน้าจอภาพ โดยใช้คำสั่งวนรอบ while และคำสั่ง break



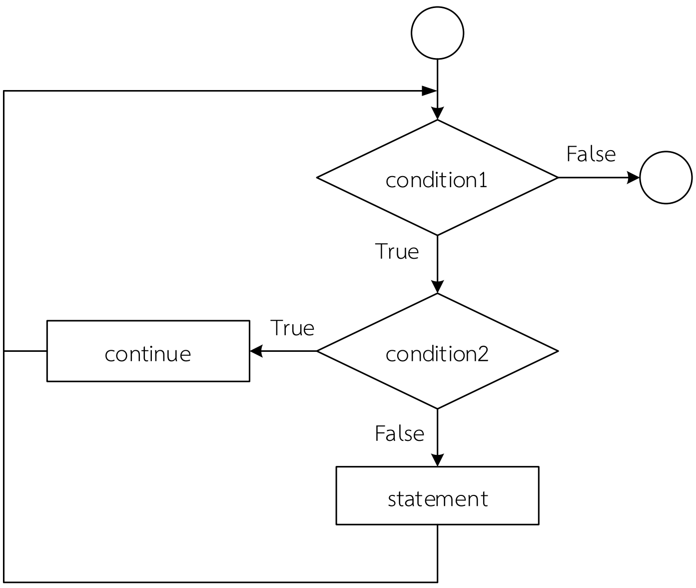
รูปที่ 8.20 ผลการทำงานของโปรแกรมแสดงเลข 1 ถึง 3 ออกหน้าจอภาพ โดยใช้คำสั่งวนรอบ while และคำสั่ง break

8.7 คำสั่ง continue

คำสั่ง continue เป็นคำสั่งที่ทำหน้าที่ควบคุมให้คำสั่งวนรอบข้ามหรือไม่ต้องทำกลุ่มคำสั่งที่อยู่ต่อจากคำสั่ง continue และเริ่มการวนรอบหรือทำซ้ำรอบถัดไปทันที นิยมใช้ข้ามหรือไม่ต้องการให้บางส่วนของกลุ่มคำสั่งทำงาน เมื่อผลการตรวจสอบเงื่อนไขที่ต้องการเป็นจริง กรณีมีรูปแบบคำสั่งดังนี้

```
continue
```

การทำงานของคำสั่ง จากแผนผังในรูปที่ 8.21 เมื่อผลการตรวจสอบเงื่อนไข condition2 เป็นจริง คำสั่ง continue จะทำงาน จากนั้นคำสั่ง continue จะทำให้การทำงานของคำสั่งวนรอบข้ามหรือไม่ให้กลุ่มคำสั่งที่อยู่ต่อจากคำสั่ง continue ทำงานและเริ่มการวนรอบหรือทำซ้ำรอบถัดไปทันที



รูปที่ 8.21 แผนผังแสดงการทำงานของคำสั่ง continue

ตัวอย่างโปรแกรมแสดงเลข 1 ถึง 5 ออกหน้าจอภาพ โดยกำหนดให้โปรแกรมไม่ต้องแสดงเลข 3 ออกหน้าจอภาพ โดยใช้คำสั่งวนรอบ while และคำสั่ง continue

```
1  #include<stdio.h>
2  int main()
3  {
4      int i;
5      i = 1;
6      while(i <= 5)
7      {
8          if(i == 3)
9          {
10             i = i + 1;
11             continue;
12         }
13         printf("%d\n",i);
14         i = i + 1;
15     }
16     return(0);
17 }
```

จากตัวอย่างโปรแกรมสามารถอธิบายการทำงานของคำสั่งบางส่วนได้ดังนี้

บรรทัดที่ 4 ประกาศให้ i เป็นตัวแปรที่มีชนิดข้อมูลเป็นเลขจำนวนเต็ม

บรรทัดที่ 5 กำหนดให้ตัวแปร i มีค่าเริ่มต้นเท่ากับ 1 (i = 1)

บรรทัดที่ 6 คำสั่งวนรอบ while เริ่มต้นทำงาน โดยทำการตรวจสอบเงื่อนไข i <= 5 ถ้าผลตรวจสอบเงื่อนไขเป็นจริง จะไปทำคำสั่งบรรทัดที่ 8 - 14 หลังจากทำคำสั่งเสร็จ จะย้อนกลับไปทำการตรวจสอบเงื่อนไข i <= 5 อีกครั้ง ทำเช่นนี้ไปเรื่อยๆ จนกว่าผลการตรวจสอบเงื่อนไข i<=5 จะเป็นเท็จ หรือตัวแปร i จะมีค่าเท่ากับ 6 การทำงานของคำสั่ง while จึงจะสิ้นสุดลง และข้ามไปทำคำสั่งบรรทัดที่ 16 (return)

บรรทัดที่ 8 คำสั่ง if ทำการตรวจเงื่อนไข i == 3 ถ้าผลตรวจสอบเงื่อนไขเป็นจริง จะไปทำคำสั่งบรรทัดที่ 10 และ 11

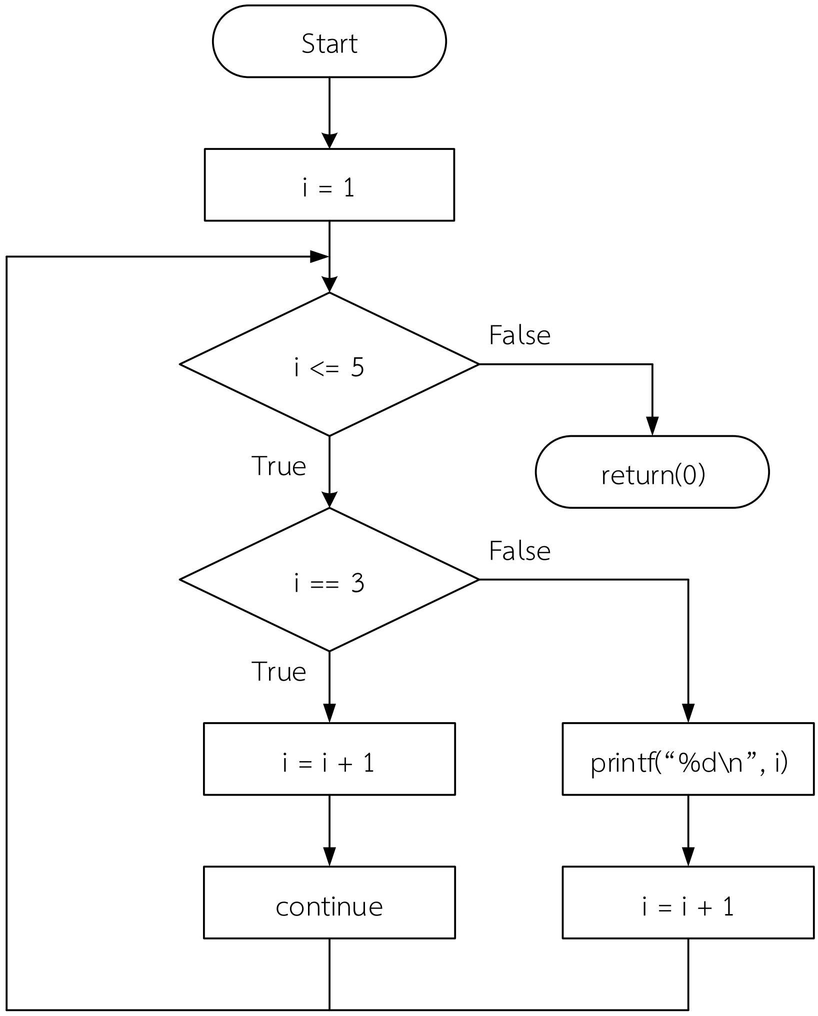
บรรทัดที่ 10 เพิ่มค่าของตัวแปร i ขึ้นเท่ากับ 1

บรรทัดที่ 11 คำสั่ง continue จะทำให้คำสั่งวนรอบ while เริ่มการวนรอบหรือทำซ้ำรอบถัดไปทันทีโดยข้ามหรือไม่ทำคำสั่งบรรทัดที่ 12 และ 13

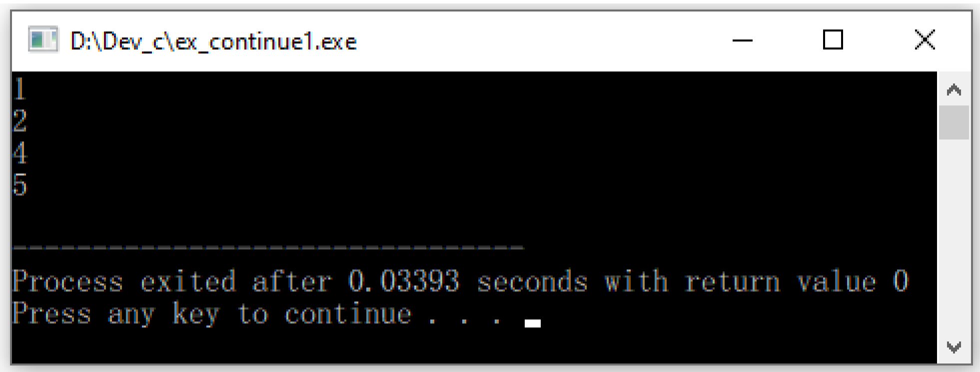
บรรทัดที่ 12 พิมพ์ค่าของตัวแปร i ออกหน้าจอภาพ

บรรทัดที่ 13 เพิ่มค่าของตัวแปร i ขึ้นเท่ากับ 1

บรรทัดที่ 16 สิ้นสุดการทำงานของโปรแกรม การทำงานของโปรแกรมแสดงดังรูปที่ 8.22 และรูปที่ 8.23



รูปที่ 8.22 แผนผังแสดงการทำงานของโปรแกรมแสดงเลข 1 ถึง 5 ออกหน้าจอภาพ โดยกำหนดให้โปรแกรมไม่ต้องแสดงเลข 3 ออกหน้าจอภาพ โดยใช้คำสั่งวนรอบ while และคำสั่ง continue



รูปที่ 8.23 ผลการทำงานของโปรแกรมแสดงเลข 1 ถึง 5 ออกหน้าจอภาพ โดยกำหนดให้โปรแกรมไม่ต้องแสดงเลข 3 ออกหน้าจอภาพ โดยใช้คำสั่งวนรอบ while และคำสั่ง continue

บทที่ 9
ฟังก์ชัน (Function)

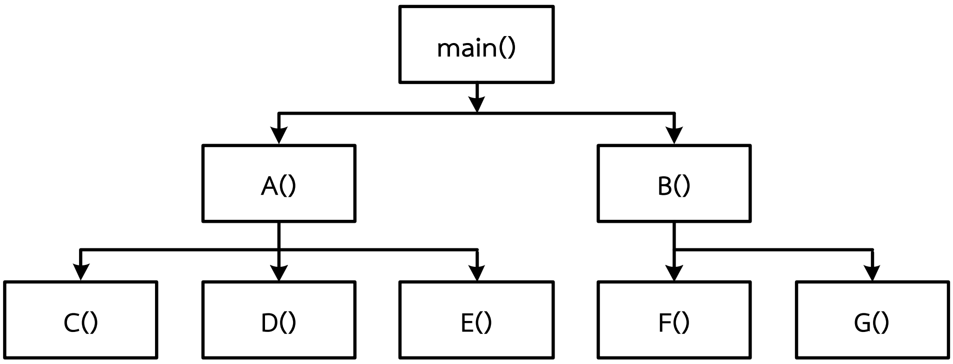
9.1 บทนำ

บทนี้กล่าวถึงการเขียนโปรแกรมเพื่อสร้างและใช้งานฟังก์ชัน ซึ่งเป็นพื้นฐานการพัฒนาโปรแกรมที่มีความซับซ้อน หลักการทำงานโดยใช้ฟังก์ชันคือการแบ่งการทำงานของโปรแกรม ที่มีขนาดใหญ่ออกเป็นส่วนย่อยและแต่ละส่วนมีหน้าที่การทำงานของตัวเอง เนื้อหาในบทนี้ประกอบด้วย (1) แนะนำฟังก์ชัน เพื่ออธิบายภาพรวมของการเขียนโปรแกรมแบบมี การสร้างและเรียกใช้งานฟังก์ชัน (2) ประเภทของฟังก์ชัน ซึ่งการสร้างฟังก์ชันสามารถแบ่งเป็น 4 ประเภท ประกอบด้วย ฟังก์ชันที่ไม่มีการคืนค่ากลับและไม่มีการรับพารามิเตอร์ ฟังก์ชันที่ไม่มี การคืนค่ากลับแต่มีการรับพารามิเตอร์ ฟังก์ชันที่มีการคืนค่ากลับแต่ไม่มีการรับพารามิเตอร์ และฟังก์ชันที่มีการคืนค่ากลับและมีการรับพารามิเตอร์ (3) ขอบเขตการทำงานของตัวแปรในฟังก์ชัน เนื่องจากตัวแปรที่ถูกสร้างในฟังก์ชันมีขอบเขตการใช้งานที่แตกต่างจากตัวแปรที่ถูกสร้างนอกฟังก์ชัน (4) ฟังก์ชันเรียกตัวเอง ฟังก์ชันประเภทนี้เป็นพื้นฐานของการดำเนินการทางคณิตศาสตร์เช่น แฟคทอเรียล เป็นต้น และ (5) ฟังก์ชันการคำนวณทางคณิตศาสตร์ เป็นการยกตัวอย่างการใช้งานฟังก์ชันมาตรฐานของภาษาซีผ่านการใช้งานฟังก์ชันการคำนวณทางคณิตศาสตร์

9.2 แนะนำฟังก์ชัน

ฟังก์ชัน (Function) คือชุดคำสั่งที่รวมกันเป็นโปรแกรมย่อย ชุดคำสั่งดังกล่าวต้องอยู่ภายในเครื่องหมาย { } ฟังก์ชันที่สร้างขึ้นต้องมีวัตถุประสงค์การทำงานที่ชัดเจน และมีการตั้งชื่อฟังก์ชันเพื่อความสะดวกในการเรียกใช้งาน ฟังก์ชันมีรูปแบบการใช้งานแตกต่างกันโดยต้องพิจารณาว่าจะมีการรับหรือไม่รับข้อมูลจากโปรแกรมที่เรียกใช้งานและจะมีการส่งหรือไม่ส่งค่าข้อมูลออกจากฟังก์ชัน รูปแบบการใช้งานของฟังก์ชันจะขึ้นอยู่กับหน้าที่และเป้าหมายการทำงานของฟังก์ชันนั้นๆ

ในการเขียนโปรแกรมภาษาซีนั้น จะเริ่มต้นการทำงานที่ฟังก์ชัน main() และฟังก์ชัน main() สามารถเรียกใช้ฟังก์ชันย่อยได้ ไม่ว่าจะเป็นฟังก์ชันที่ผู้ใช้สร้างขึ้น (User-Defined Function) หรือฟังก์ชันมาตรฐานของภาษาซี (Standard Library Function) นอกจากนี้ฟังก์ชันย่อยยังสามารถเรียกใช้ฟังก์ชันย่อยอื่นได้เช่นกัน ตัวอย่างเช่น (ดูรูปที่ 9.1 ประกอบ) ฟังก์ชัน main() เรียกใช้งานฟังก์ชัน A และฟังก์ชัน B จากนั้นฟังก์ชัน A เรียกใช้ฟังก์ชัน C ฟังก์ชัน D และฟังก์ชัน E ส่วนฟังก์ชัน B เรียกใช้ฟังก์ชัน F และฟังก์ชัน G เป็นต้น



รูปที่ 9.1 การเรียกใช้งานฟังก์ชัน

9.3 ประเภทของฟังก์ชัน

รูปแบบของฟังก์ชันในการเขียนโปรแกรมภาษาซีแบ่งเป็น 4 รูปแบบ ประกอบด้วย

- ฟังก์ชัน ที่ไม่มีการคืนค่ากลับและไม่มีการรับพารามิเตอร์ (Void Functions without Parameters)
- ฟังก์ชัน ที่ไม่มีการคืนค่ากลับแต่มีการรับพารามิเตอร์ (Void Functions with Parameters)
- ฟังก์ชัน ที่มีการคืนค่ากลับแต่ไม่มีการรับพารามิเตอร์ (Functions Return Value without Parameters)
- ฟังก์ชันที่มีการคืนค่ากลับและมีการรับพารามิเตอร์ (Functions Return Value with Parameters)

ฟังก์ชัน แต่ละประเภทมีรายละเอียดดังนี้

9.3.1 ฟังก์ชันที่ไม่มีการคืนค่ากลับและไม่มีการรับพารามิเตอร์

ฟังก์ชันที่ไม่มีการคืนค่ากลับและไม่มีการรับพารามิเตอร์ (Void Function without Parameters) เป็นฟังก์ชันที่สร้างขึ้นโดยไม่มีการรับข้อมูล (พารามิเตอร์) จากฟังก์ชันที่เรียกใช้งาน และเมื่อฟังก์ชันทำงานเสร็จจะไม่มีการคืนค่าข้อมูลกลับไปให้ฟังก์ชันที่เรียกใช้งาน ซึ่งมีรูปแบบ การสร้างฟังก์ชันดังตัวอย่างที่ 9.1

ตัวอย่างที่ 9.1 รูปแบบการสร้างฟังก์ชันที่ไม่มีการคืนค่ากลับและไม่มีการรับพารามิเตอร์

```
void functionName (void)
{
     statements;
}
```

โดย คีย์เวิร์ด void หน้าชื่อฟังก์ชันเป็นการกำหนดว่าฟังก์ชันนั้นไม่มีการคืนค่ากลับ ไปยังฟังก์ชันที่เรียกใช้งาน functionName คือชื่อฟังก์ชันที่ต้องการสร้าง คีย์เวิร์ด void หลังชื่อฟังก์ชันเป็นการกำหนดว่าฟังก์ชันนั้นไม่มีการรับพารามิเตอร์จากฟังก์ชันที่เรียกใช้งาน และ statements คือชุดคำสั่งภายในฟังก์ชัน

ตัวอย่างที่ 9.2 การใช้งานฟังก์ชันที่ไม่มีการคืนค่ากลับและไม่มีการรับพารามิเตอร์

```
1   #include<stdio.h>
2   void PrintName(void)
3   {
4       printf("My name is CPE.");
5   }
6
7   void main()
8   {
9       PrintName();
10  }
```

ผลลัพธ์ตัวอย่างที่ 9.2

```
My name is CPE.
```

คำอธิบายตัวอย่างที่ 9.2

บรรทัดที่ 2 ถึง 5	สร้างฟังก์ชัน PrintName() สำหรับการแสดงผลข้อความ เป็นฟังก์ชันที่ไม่มีการคืนค่ากลับและไม่มีการรับพารามิเตอร์
บรรทัดที่ 7 ถึง 10	สร้างฟังก์ชัน main() เป็นฟังก์ชันหลักของการเขียนโปรแกรมภาษาซี และเป็นฟังก์ชันเริ่มต้นการทำงานของโปรแกรม
บรรทัดที่ 9	เรียกใช้งานฟังก์ชัน PrintName() ที่ไม่มีการคืนค่ากลับและไม่มี การรับพารามิเตอร์

9.2.2 ฟังก์ชันที่ไม่มีการคืนค่ากลับแต่มีการรับพารามิเตอร์

ฟังก์ชันที่ไม่มีการคืนค่ากลับแต่มีการรับพารามิเตอร์ (Void Functions with Parameters) เป็นฟังก์ชันที่สร้างขึ้นโดยมีการรับค่าข้อมูล (พารามิเตอร์) จากฟังก์ชันที่เรียกใช้งานและเมื่อฟังก์ชันทำงานเสร็จจะไม่มีการคืนค่าข้อมูลกลับไปให้ฟังก์ชันที่เรียกใช้งาน ซึ่งมีรูปแบบการสร้างฟังก์ชัน ดังตัวอย่างที่ 9.3

ตัวอย่างที่ 9.3 รูปแบบการสร้างฟังก์ชันที่ไม่มีการคืนค่ากลับแต่มีการรับพารามิเตอร์

```
void functionName (typeParameter_1 varName_1,…,typeParameter_n varName_n)
{
    statements;
}
```

โดยคีย์เวิร์ด void หน้าชื่อฟังก์ชันเป็นการกำหนดว่าฟังก์ชันนั้นไม่มีการคืนค่ากลับ ไปยังฟังก์ชันที่เรียกใช้งาน functionName คือชื่อฟังก์ชันที่ต้องการสร้าง typeParameter คือข้อมูลชนิดที่ต้องการรับจากฟังก์ชันที่เรียกใช้งาน varName_n คือชื่อตัวแปรที่ใช้รับข้อมูลจากฟังก์ชัน ที่เรียกใช้งาน และ statements คือ ชุดคำสั่งภายในฟังก์ชัน

ฟังก์ชันที่มีการรับค่าสามารถแบ่งย่อยได้เป็นประเภทต่างๆ ตามลักษณะของพารามิเตอร์ที่ส่งให้ฟังก์ชัน ดังนี้

- Pass by Value ฟังก์ชันจะรับค่าตัวแปร ค่าคงที่ หรือนิพจน์ โดยการเปลี่ยนแปลงของตัวแปรภายในจะไม่มีผลกับตัวแปรด้านนอก ถึงแม้ตัวแปรจะมีชื่อเดียวกัน
- Pass by Reference ฟังก์ชันจะรับตำแหน่งในหน่วยความจำของตัวแปร หากมีการเปลี่ยนแปลงค่าภายในฟังก์ชัน ตัวแปรที่อยู่นอกฟังก์ชันจะเปลี่ยนแปลงด้วย เนื่องจากเป็นตัวแปรที่ใช้หน่วยความจำตำแหน่งเดียวกัน

นอกจากนี้ยังมีการส่ง**อาร์เรย์เป็นพารามิเตอร์**ให้กับฟังก์ชัน ซึ่งจะถือเป็นแบบ Pass by Reference

ตัวอย่างที่ 9.4 การใช้งานฟังก์ชันที่ไม่มีการคืนค่ากลับแต่มีการรับพารามิเตอร์แบบ Pass by Value (1)

```
#include<stdio.h>
void Multiplication(int a, int b)
{
    int answer;
    answer = a * b;
    printf("Multiplication of %d * %d = %d",a, b, answer);
}
void main()
{
    int x = 4, y = 5;
    Multiplication(x, y);
}
```

ผลลัพธ์ตัวอย่างที่ 9.4

```
Multiplication of 4 * 5 = 20
```

คำอธิบายตัวอย่างที่ 9.4

บรรทัดที่ 2 ถึง 7	สร้างฟังก์ชัน Multiplication() เป็นฟังก์ชันที่ไม่มีการคืนค่ากลับแต่มีการรับพารามิเตอร์ 2 ตัวเป็นข้อมูลชนิดจำนวนเต็ม คือตัวแปร a รับค่าข้อมูลจากตัวแปร x และตัวแปร b รับค่าข้อมูลจากตัวแปร y
บรรทัดที่ 8 ถึง 12	สร้างฟังก์ชัน main() เป็นฟังก์ชันหลักของการเขียนโปรแกรมภาษาซี และเป็นฟังก์ชันเริ่มต้นการทำงานของโปรแกรม
บรรทัดที่ 11	เรียกใช้งานฟังก์ชัน Multiplication() โดยส่งข้อมูลอาร์กิวเมนต์ 2 ตัว เป็นข้อมูลชนิดจำนวนเต็ม คือตัวแปร x และ y

ตัวอย่างที่ 9.5 การใช้งานฟังก์ชันที่ไม่มีการคืนค่ากลับแต่มีการรับพารามิเตอร์แบบ Pass by Value (2)

```c
#include<stdio.h>

void MaxData(int num1, int num2)
{
    if(num1 == num2 )
    {
        printf("Data %d = %d", num1, num2);
    }
    else if(num1 > num2)
    {
        printf("Data max = %d", num1);
    }
    else
    {
        printf("Data max = %d", num2);
    }
}

void main()
{
    int a = 10, b = 7;
    printf("Data max checking of a = %d and b = %d \n", a, b);
    MaxData(a, b);
}
```

ผลลัพธ์ตัวอย่างที่ 9.5

```
Data max checking of a = 10 and b = 7
Data max = 10
```

คำอธิบายตัวอย่างที่ 9.5

บรรทัดที่ 3 ถึง 17	สร้างฟังก์ชัน DataMax() สำหรับการหาค่าสูงสุด เป็นฟังก์ชันที่ไม่มีการคืนค่ากลับแต่มีการรับพารามิเตอร์ 2 ตัวเป็นข้อมูลชนิดจำนวนเต็ม
บรรทัดที่ 19 ถึง 24	สร้างฟังก์ชัน main() เป็นฟังก์ชันหลักของการเขียนโปรแกรมภาษาซี และเป็นฟังก์ชันเริ่มต้นการทำงานของโปรแกรม
บรรทัดที่ 23	เรียกใช้งานฟังก์ชัน MaxData() ที่ไม่มีการคืนค่ากลับแต่มีการรับพารามิเตอร์ โดยส่งข้อมูลอาร์กิวเมนต์ 2 ตัว เป็นข้อมูลชนิดจำนวนเต็ม คือตัวแปร a และ b

ตัวอย่างที่ 9.6 การใช้งานฟังก์ชันที่ไม่มีการคืนค่ากลับแต่มีการรับพารามิเตอร์แบบ Pass by Reference

```c
#include<std#include <stdio.h>

void NumChange(int *number2)
{
   printf("Before change : %d\n",*number2);
   *number2 += 5;
   printf("After change  : %d\n",*number2);
}

void main( )
{
   int number = 15;
   NumChange(&number);
   printf("After entering the function : %d\n",number);
}
```

ผลลัพธ์ตัวอย่างที่ 9.6

```
Before change : 15
After change  :  20
After entering the function :  20
```

คำอธิบายตัวอย่างที่ 9.6

บรรทัดที่ 3 ถึง 8	สร้างฟังก์ชัน NumChange( ) สำหรับการคำนวณเพิ่มค่า 5 เป็นฟังก์ชันที่ไม่มีการคืนค่ากลับแต่มีการรับพารามิเตอร์เป็นข้อมูล pointer ชนิดจำนวนเต็ม ซึ่ง ตัวแปรที่มี * นำหน้าชื่อตัวแปรจะเป็นเก็บตำแหน่งของข้อมูล
บรรทัดที่ 10 ถึง 15	สร้างฟังก์ชัน main() เป็นฟังก์ชันหลักของการเขียนโปรแกรมภาษาซี และเป็นฟังก์ชันเริ่มต้นการทำงานของโปรแกรม
บรรทัดที่ 13	เรียกใช้งานฟังก์ชัน NumChange( ) ที่ไม่มีการคืนค่ากลับแต่มีการรับพารามิเตอร์ โดยส่งข้อมูลอาร์กิวเมนต์ 1 ตัว เป็นตำแหน่ง (&) ข้อมูลชนิดจำนวนเต็ม คือ ตัวแปร number

ตัวอย่างที่ 9.7 การใช้งานฟังก์ชันที่ไม่มีการคืนค่ากลับแต่มีการรับพารามิเตอร์เป็นอาร์เรย์ (Pass by Reference)

```
#include <stdio.h>

void Dollar(float baht[ ], int num)
{
   int i;
   printf("Before changing : %.02f\n", baht[0]);
   for(i=0; i<num; i++)
      baht[i] *= 40.0;
   printf("After changing  : %.02f\n", baht[0]);
}

void main()
{
   float money[3] = {200.0, 400.0, 600.0};
   Dollar(money, 3);
   printf("After entering function: %.02f\n", money[0]);
}
```

ผลลัพธ์ตัวอย่างที่ 9.7

```
Before changing : 200.00
After changing  : 8000.00
After entering the function :  8000.00
```

คำอธิบายตัวอย่างที่ 9.7

บรรทัดที่ 3 ถึง 10 สร้างฟังก์ชัน Dollar( ) สำหรับการคำนวณเปลี่ยนสกุลเงิน เป็นฟังก์ชันที่ไม่มีการคืนค่ากลับแต่มีการรับพารามิเตอร์เป็นข้อมูลอาร์เรย์ ชนิดจำนวนเต็ม ซึ่งรับค่าขนาดและตำแหน่งของตัวแปรอาร์เรย์ ตามจำนวนของอาร์กิวเมนต์ที่ส่งมา

บรรทัดที่ 12 ถึง 17 สร้างฟังก์ชัน main( ) เป็นฟังก์ชันหลักของการเขียนโปรแกรมภาษาซี และเป็นฟังก์ชันเริ่มต้นการทำงานของโปรแกรม

บรรทัดที่ 15 เรียกใช้งานฟังก์ชัน Dollar( ) ที่ไม่มีการคืนค่ากลับแต่มีการรับพารามิเตอร์ โดยส่งข้อมูลอาร์กิวเมนต์ 2 ตัว ซึ่ง money หรือ &money[0] เป็นการส่งตำแหน่งของข้อมูลในอาร์เรย์ตัวแรกและ ข้อมูลชนิดจำนวนเต็ม คือ 3

9.2.3 ฟังก์ชันที่มีการคืนค่ากลับแต่ไม่มีการรับพารามิเตอร์

ฟังก์ชันที่มีการคืนค่ากลับแต่ไม่มีการรับพารามิเตอร์ (Function Return without Parameters) เป็นฟังก์ชันที่สร้างขึ้นโดยไม่มีการรับค่าข้อมูล (พารามิเตอร์) จากฟังก์ชันที่เรียกใช้งาน และ เมื่อฟังก์ชันทำงานเสร็จจะมีการคืนค่าข้อมูลกลับไปให้ฟังก์ชันที่เรียกใช้งาน ซึ่งมีรูปแบบการสร้างฟังก์ชันดังตัวอย่างที่ 9.8

ตัวอย่างที่ 9.8 รูปแบบการสร้างฟังก์ชันที่มีการคืนค่ากลับแต่ไม่มีการรับพารามิเตอร์

```
typeReturn functionName (void)
{
      statements;
      return varNameReturn;
 }
```

โดยที่ typeReturn คือชนิดข้อมูลที่ต้องการคืนค่ากลับไปยังฟังก์ชันที่เรียกใช้งาน functionName คือชื่อฟังก์ชันที่ต้องการสร้าง คีย์เวิร์ด void หลังชื่อฟังก์ชันเป็นการกำหนดว่าฟังก์ชันนั้นไม่มีการรับพารามิเตอร์จากฟังก์ชันที่เรียกใช้งาน statements คือชุดคำสั่งภายในฟังก์ชัน varNameReturn คือชื่อตัวแปรที่คืนค่ากลับให้ฟังก์ชันที่เรียกใช้งานซึ่งมีชนิดข้อมูลแบบเดียวกับ typeReturn

ตัวอย่างที่ 9.9 การใช้งานฟังก์ชันที่มีการคืนค่ากลับแต่ไม่มีการรับพารามิเตอร์

```
1  #include<stdio.h>
2
3  int Multiplication(void)
4  {
5      int a, b, c;
6      a = 4;
7      b = 5;
8      c = a * b;
9      return c;
10 }
11
12 void main()
13 {
14     int answer;
15     answer = Multiplication();
16     printf("Answer = %d", answer);
17 }
```

ผลลัพธ์ตัวอย่างที่ 9.9

```
Answer = 20;
```

คำอธิบายตัวอย่างที่ 9.9

บรรทัดที่ 3 ถึง 10	สร้างฟังก์ชัน Multiplication() สำหรับการหาผลคูณ เป็นฟังก์ชันที่มีการคืนค่ากลับแต่ไม่มีการรับพารามิเตอร์
บรรทัดที่ 12 ถึง 17	สร้างฟังก์ชัน main() เป็นฟังก์ชันหลักของการเขียนโปรแกรมภาษาซี และเป็นฟังก์ชันเริ่มต้นการทำงานของโปรแกรม
บรรทัดที่ 15	เรียกใช้งานฟังก์ชัน Multiplication() ที่มีการคืนค่ากลับแต่ไม่มีการรับพารามิเตอร์ และให้ตัวแปร answer รับค่าผลลัพธ์จากฟังก์ชัน

9.2.4 ฟังก์ชันที่มีการคืนค่ากลับและมีการรับพารามิเตอร์

ฟังก์ชันที่มีการคืนค่ากลับและมีการรับพารามิเตอร์ (Function Return Value with Parameters) เป็นฟังก์ชันที่สร้างขึ้นโดยมีการรับค่าข้อมูล (พารามิเตอร์) จากฟังก์ชันที่เรียกใช้งาน และเมื่อฟังก์ชันทำงานเสร็จจะมีการคืนค่าข้อมูลกลับไปให้ฟังก์ชันที่เรียกใช้งาน ซึ่งมีรูปแบบการสร้างฟังก์ชันดังตัวอย่างที่ 9.10

ตัวอย่างที่ 9.10 รูปแบบการสร้างฟังก์ชันที่มีการคืนค่ากลับแต่ไม่มีการรับพารามิเตอร์

```
typeReturn functionName (typeParameter_1 varName_1,...,typeParameter_n varName_n)
{
      statements;
      return  varNameReturn;
}
```

โดย	typeReturn	คือชนิดข้อมูลที่ต้องการคืนค่ากลับให้ฟังก์ชันที่เรียกใช้งาน
	functionName	คือชื่อฟังก์ชันที่ต้องการสร้าง
	typeParameter_n	คือชนิดข้อมูลที่ต้องการรับจากฟังก์ชันที่เรียกใช้งาน
	varName_n	คือเป็นชื่อตัวแปรที่ใช้รับข้อมูลจากฟังก์ชันที่เรียกใช้งาน
	statements	คือชุดคำสั่งภายในฟังก์ชัน
	varNameReturn	คือชื่อตัวแปรที่คืนค่ากลับให้ฟังก์ชันที่เรียกใช้งานซึ่งมีชนิดข้อมูลประเภทเดียวกับ typeReturn

ตัวอย่างที่ 9.11 การใช้งานฟังก์ชันที่มีการคืนค่ากลับและมีการรับพารามิเตอร์ (1)

```c
#include<stdio.h>
#include<math.h>

float Area_of_Circle(int r)
{
    float a;
    a = M_PI * r * r;
    return a;
}
void main()
{
    int radius = 8;
    float area;
    area = Area_of_Circle(radius);
    printf("Area of Circle = %f", area);
}
```

ผลลัพธ์ตัวอย่างที่ 9.11

```
Area of Circle = 200.960007
```

คำอธิบายตัวอย่างที่ 9.11

บรรทัดที่ 2	คำสั่งเรียกใช้ฟังก์ชันมาตรฐานไฟล์ math.h ที่รวบรวมฟังก์ชันและค่าคงที่ต่าง ๆ ทางคณิตศาสตร์
บรรทัดที่ 3 ถึง 8	สร้างฟังก์ชัน Area_of_Circle() สำหรับการหาพื้นที่ของวงกลม เป็นฟังก์ชัน ที่มีการคืนค่ากลับและมีการรับพารามิเตอร์
บรรทัดที่ 7	คำนวณหาค่าของ a โดย เรียกใช้ค่าคงที่ M_PI จาก math.h
บรรทัดที่ 10 ถึง 16	สร้างฟังก์ชัน main() เป็นฟังก์ชันหลักของการเขียนโปรแกรมภาษาซี และเป็นฟังก์ชันเริ่มต้นการทำงานของโปรแกรม
บรรทัดที่ 14	เรียกใช้งานฟังก์ชัน Area_of_Circle() ที่มีการคืนค่ากลับและมีการรับพารามิเตอร์ และให้ตัวแปร area รับค่าผลลัพธ์จากฟังก์ชัน

ตัวอย่างที่ 9.12 การใช้งานฟังก์ชันที่มีการคืนค่ากลับและมีการรับพารามิเตอร์ (2)

```c
#include<stdio.h>

int MinData(int num1, int num2)
{
    int minimum;

    if(num1 <= num2)
    {
        minimum = num1;
    }
    else
    {
        minimum = num2;
    }
    return minimum;
}
void main()
{
    int a = 15, b = 7;
    printf("Data min checking of a = %d and b = %d \n", a,b);
    printf("Data min = %d", MinData(a,b));
}
```

ผลลัพธ์ตัวอย่างที่ 9.12

```
Data min checking of a = 15 and b = 7
Data min = 7
```

คำอธิบายตัวอย่างที่ 9.12

บรรทัดที่ 3 ถึง 16 สร้างฟังก์ชัน MinData() สำหรับการหาค่าต่ำสุด เป็นฟังก์ชันที่มีการคืนค่ากลับและมีการรับพารามิเตอร์

บรรทัดที่ 17 ถึง 22 สร้างฟังก์ชัน main() เป็นฟังก์ชันหลักของการเขียนโปรแกรมภาษาซี และเป็นฟังก์ชันเริ่มต้นการทำงานของโปรแกรม

บรรทัดที่ 21 เรียกใช้งานฟังก์ชัน MinData() ที่มีการคืนค่ากลับและมีการรับพารามิเตอร์ จำนวน 2 ตัว เป็นจำนวนเต็มและคืนค่าเป็นข้อมูลชนิดจำนวนเต็ม

9.4 ขอบเขตการทำงานของตัวแปรในฟังก์ชัน

การใช้งานตัวแปรที่ถูกสร้างขึ้นในฟังก์ชันจะมีขอบเขตการใช้งานแตกต่างกับตัวแปรที่ถูกสร้างขึ้นนอกฟังก์ชัน ในการเขียนโปรแกรมภาษาซีตัวแปรที่มีลักษณะแตกต่างกันดังกล่าวเรียกว่าตัวแปรแบบโลคอล และ ตัวแปรแบบโกลบอล โดยที่ ตัวแปรแบบโลคอล เป็นตัวแปรที่มีการประกาศ ใช้งานภายในฟังก์ชันจึงไม่สามารถเรียกใช้งานภายนอกฟังก์ชันได้ และ ตัวแปรแบบโกลบอล เป็นตัวแปรที่มีการประกาศใช้งานตั้งแต่เริ่มต้นโปรแกรมจึงสามารถเรียกใช้งานในส่วนใดของโปรแกรมก็ได้ ศึกษาตัวอย่างที่ 9.13 เพื่อความเข้าใจเรื่องขอบเขตการทำงานของตัวแปรในฟังก์ชัน

ตัวอย่างที่ 9.13 ขอบเขตการทำงานของตัวแปรในฟังก์ชัน

```c
#include<stdio.h>

int a = 8;

void TestVariables(int b)
{
    a = a + 5;
    b = b + 5;
    printf("Printing from TestVariable() \n");
    printf("a = %d, b = %d \n\n", a,b);
}
void main()
{
    int b = 4;
    printf("Printing from main() before run TestVariable() \n");
    printf("a = %d, b = %d \n\n", a,b);
    TestVariables(b);
    printf("Printing from main() after run TestVariable() \n");
    printf("a = %d, b = %d \n\n", a,b);
}
```

ผลลัพธ์ตัวอย่างที่ 9.13

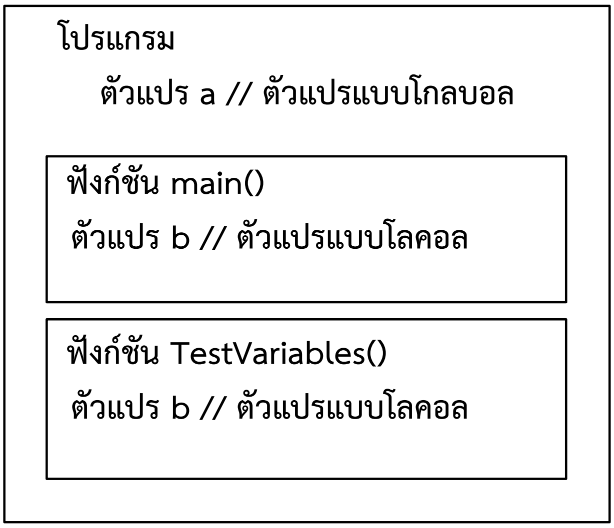
```
Printing from main() before run TestVariable()
a = 8, b = 4

Printing from TestVariable()
a = 13, b = 9

Printing from main() after run TestVariable()
a = 13, b = 4
```

คำอธิบายตัวอย่างที่ 9.13

จากโปรแกรมตัวอย่างที่ 9.13 สามารถเขียนแผนผังขอบเขตของตัวแปรได้ดังรูปที่ 9.2



รูปที่ 9.2 ขอบเขตการทำงานของตัวแปรในฟังก์ชัน

บรรทัดที่ 3	ประกาศตัวแปร a เป็นตัวแปรแบบโกลบอลและกำหนดให้เก็บค่าคงที่ 8 ซึ่งสามารถเรียกใช้งานได้จากทุกส่วนในโปรแกรม
บรรทัดที่ 5 ถึง 11	สร้างฟังก์ชัน TestVariables() เป็นฟังก์ชันที่ไม่มีการคืนค่ากลับแต่มีการรับพารามิเตอร์ 1 ตัว เป็นข้อมูลประเภทจำนวนเต็ม จากตัวอย่างที่ 9.11กำหนดให้ตัวแปร b รับข้อมูลจากฟังก์ชัน main() ซึ่งสามารถเรียกใช้งาน ได้เฉพาะภายในฟังก์ชัน TestVariables() เท่านั้น จากผลลัพธ์ของโปรแกรม ที่ 9.11 พบว่า ตัวแปร b ในฟังก์ชัน main() เป็นคนละตัวกับตัวแปร b ในฟังก์ชัน TestVariables()

บรรทัดที่ 12 ถึง 20	สร้างฟังก์ชัน main() เป็นฟังก์ชันหลักของการเขียนโปรแกรมภาษาซี และเป็นฟังก์ชันเริ่มต้นการทำงานของโปรแกรม
บรรทัดที่ 14	ประกาศตัวแปร b เป็นตัวแปรแบบโลคอล และกำหนดให้เก็บค่าคงที่ 4 ซึ่งสามารถเรียกใช้งานได้เฉพาะในส่วนของฟังก์ชัน main() เท่านั้น
บรรทัดที่ 17	เรียกใช้งานฟังก์ชัน TestVariables() ที่ไม่มีการคืนค่ากลับแต่มีการรับพารามิเตอร์จำนวน 1 ตัว เป็นจำนวนเต็ม จากตัวอย่างที่ 9.11 คือการรับพารามิเตอร์ b ของฟังก์ชัน TestVariables()

9.5 ฟังก์ชันเรียกตัวเอง

หัวข้อที่ผ่านมาจะเป็นการใช้งานฟังก์ชันในลักษณะที่ฟังก์ชันหนึ่งเรียกใช้งานฟังก์ชันตัวอื่น แต่ในหัวข้อนี้จะเป็นการศึกษาเรื่องฟังก์ชันเรียกตัวเอง (Recursive Function) คือการที่ฟังก์ชันใดฟังก์ชันหนึ่งเรียกให้ตัวเองทำงาน ศึกษาตัวอย่างที่ 9.14 เพื่อความเข้าใจเรื่องฟังก์ชันเรียกตัวเอง

ตัวอย่างที่ 9.14 การใช้งานฟังก์ชันเรียกตัวเอง

```
#include<stdio.h>

Recursive(int n)
{
    if(n<=1)
    {
        return 1;
    }
    else
    {
        return(n * Recursive(n-1));
    }
}

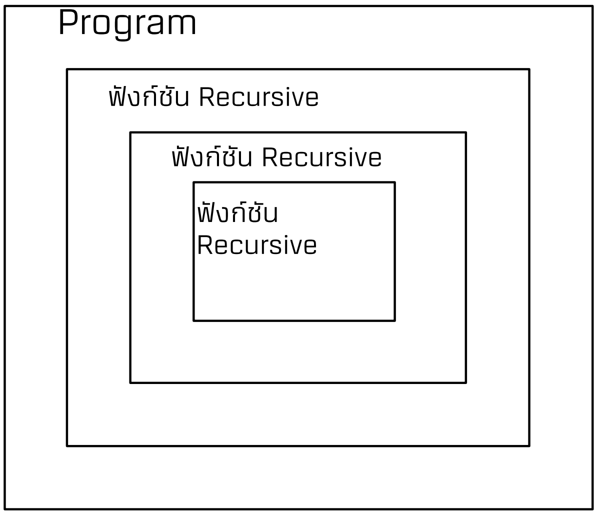
void main()
{
    int a = 6;
    printf(" 6! = %d", Recursive(a));
}
```

ผลลัพธ์ตัวอย่างที่ 9.14

```
6! = 720
```

คำอธิบายตัวอย่างที่ 9.14

จากโปรแกรมตัวอย่างที่ 9.14 สามารถเขียนแผนผังขอบเขตของตัวแปรได้ดังรูปที่ 9.3



รูปที่ 9.3 ฟังก์ชันเรียกตัวเอง

บรรทัดที่ 3 ถึง 13	สร้างฟังก์ชัน Recusive() เพื่อหาค่าแฟคทอเรียล เป็นฟังก์ชันที่มีการคืนค่ากลับและมีการรับพารามิเตอร์ 1 ตัว เป็นข้อมูลประเภทจำนวนเต็ม และคืนค่าข้อมูลประเภทจำนวนเต็ม
บรรทัดที่ 15 ถึง 19	สร้างฟังก์ชัน main() เป็นฟังก์ชันหลักของการเขียนโปรแกรมภาษาซี และเป็นฟังก์ชันเริ่มต้นการทำงานของโปรแกรม
บรรทัดที่ 18	เรียกใช้งานฟังก์ชัน Recursive() ที่มีการคืนค่ากลับและมีการรับพารามิเตอร์ และแสดงผลลัพธ์ทางจอภาพ

9.6 ฟังก์ชันมาตรฐาน (Standard Library Functions)

ในหัวข้อที่ผ่านมาอธิบายถึงการสร้างและใช้งานฟังก์ชันที่ผู้ใช้เป็นผู้สร้างและกำหนดให้ฟังก์ชันทำงานตามความต้องการ แต่หัวข้อนี้จะอธิบายการเรียกใช้งานฟังก์ชันมาตรฐานของภาษาซี โดยยกตัวอย่างการใช้งานฟังก์ชันการคำนวณทางคณิตศาสตร์เพื่อเป็นแนวทางการใช้งานฟังก์ชันมาตรฐานและเป็นพื้นฐานในการประยุกต์ใช้งานกับฟังก์ชันมาตรฐานอื่นได้

9.5.1 ฟังก์ชันการคำนวณทางคณิตศาสตร์

การคำนวณทางคณิตสาสตร์เป็นฟังก์ชันมาตรฐานที่มีการใช้งานบ่อยครั้ง ฟังก์ชันดังกล่าวอยู่ในไลบารี math.h โดยจะยกตัวอย่างเฉพาะฟังก์ชันที่น่าสนใจและใช้งานบ่อยครั้ง รายละเอียด ดังตารางที่ 9.1

ตารางที่ 9.1 ฟังก์ชันมาตรฐานสำหรับการคำนวณทางคณิตศาสตร์

ฟังก์ชัน	คำอธิบาย
sin(x)	ฟังก์ชันหาค่า sin ของมุม โดย x เป็นมุมที่ต้องการหามีหน่วยเป็นเรเดียน
cos(x)	ฟังก์ชันหาค่า cos ของมุม โดย x เป็นมุมที่ต้องการหามีหน่วยเป็นเรเดียน
tan(x)	ฟังก์ชันหาค่า tan ของมุม โดยที่ x เป็นมุมที่ต้องการหา มีหน่วยเป็นเรเดียน
sqrt(x)	ฟังก์ชันหารากที่สอง โดย x เป็นตัวแปรหรือค่าคงที่จำนวนเต็มบวกหรือศูนย์
pow(x,y)	ฟังก์ชันหาค่ายกกำลัง โดย x เป็นตัวแปรหรือค่าคงที่เลขฐานจำนวนเต็มบวกหรือศูนย์
log(x)	ฟังก์ชันหาค่า log ฐาน e โดย x เป็นตัวแปรหรือค่าคงที่จำนวนเต็มบวกหรือศูนย์
log10(x)	ฟังก์ชันหาค่า log ฐาน 10 โดย x เป็นตัวแปรหรือค่าคงที่จำนวนเต็มบวกหรือศูนย์
fabs(x)	ฟังก์ชันหาค่าสัมบูรณ์ โดย x เป็นตัวแปรหรือค่าคงที่

ตัวอย่างที่ 9.15 การใช้งานฟังก์ชันการคำนวณทางคณิตศาสตร์

```c
1   #include<stdio.h>
2   #include<conio.h>
3   #include<math.h>
4
5   void main()
6   {
7       int x = 90, y = 5;
8       printf("x = 90 \n");
9       printf("sin(x)      = %f \n", sin(x));
10      printf("cos(x)      = %f \n", cos(x));
11      printf("tan(x)      = %f \n", tan(x));
12      printf("sqrt(x)     = %f \n", sqrt(x));
13      printf("pow(x,y)    = %f \n", pow(x,y));
14      printf("log(x)      = %f \n", log(x));
15      printf("log10(x)    = %f \n", log10(x));
16      printf("fabs(x)     = %f \n", fabs(x));
17  }
```

ผลลัพธ์ตัวอย่างที่ 9.15

```
x = 90
sin(x)     = 0.893997
cos(x)     = -0.448074
tan(x)     = -1.995200
sqrt(x)    = 9.486833
pow(x,y)   = 5904900000.000000
log(x)     = 4.499810
log10(x)   = 1.954243
fabs(x)    = 90.000000
```

คำอธิบายตัวอย่างที่ 9.15

บรรทัดที่ 10	แสดงค่า sin ของมุม x
บรรทัดที่ 11	แสดงค่า cos ของมุม x
บรรทัดที่ 12	แสดงค่า tan ของมุม x
บรรทัดที่ 13	แสดงค่าหาค่ารากที่สองของ x
บรรทัดที่ 14	แสดงหาค่ายกกำลัง xy
บรรทัดที่ 15	แสดงค่า log ฐาน e
บรรทัดที่ 16	แสดงค่า log ฐาน 10
บรรทัดที่ 17	แสดงค่าสัมบูรณ์

สรุปเนื้อหาบทที่ 9

ฟังก์ชัน (Function) คือชุดคำสั่งที่รวมกันเป็นโปรแกรมย่อย ชุดคำสั่งดังกล่าวต้องอยู่ภายในเครื่องหมาย { } ฟังก์ชันที่สร้างขึ้นต้องมีวัตถุประสงค์การทำงานที่ชัดเจน และมีการตั้งชื่อฟังก์ชัน เพื่อความสะดวกในการเรียกใช้งาน ฟังก์ชันมีรูปแบบการใช้งานแตกต่างกันโดยต้องพิจารณา ว่าจะมีการรับหรือไม่รับข้อมูลจากโปรแกรมที่เรียกใช้งานและจะมีการส่งหรือไม่ส่งค่าข้อมูลออกจากฟังก์ชัน รูปแบบการใช้งานของฟังก์ชันจะขึ้นอยู่กับหน้าที่และเป้าหมายการทำงานของฟังก์ชันนั้นๆ โดยรูปแบบของฟังก์ชันในการเขียนโปรแกรมภาษาซีแบ่งเป็น 4 รูปแบบ ประกอบด้วย **ฟังก์ชันที่ไม่มีการคืนค่ากลับและไม่มีการรับพารามิเตอร์** (Void Functions without Parameters) เป็นฟังก์ชัน ที่สร้างขึ้นโดยไม่มีการรับข้อมูล (พารามิเตอร์) จากฟังก์ชันที่เรียกใช้งาน และเมื่อฟังก์ชันทำงานเสร็จจะไม่มีการคืนค่าข้อมูลกลับไปให้ฟังก์ชันที่เรียกใช้งาน **ฟังก์ชันที่ไม่มีการคืนค่ากลับแต่มีการรับพารามิเตอร์** (Void Functions with Parameters) เป็นฟังก์ชันที่สร้างขึ้นโดยมีการรับค่าข้อมูล (พารามิเตอร์) จากฟังก์ชันที่เรียกใช้งานและเมื่อฟังก์ชันทำงานเสร็จจะไม่มีการคืนค่าข้อมูลกลับไป ให้ฟังก์ชันที่เรียกใช้งาน **ฟังก์ชันที่มีการคืนค่ากลับแต่ไม่มีการรับพารามิเตอร์** (Functions Return Value without Parameters) เป็นฟังก์ชันที่สร้างขึ้นโดยไม่มีการรับค่าข้อมูล (พารามิเตอร์) จากฟังก์ชันที่เรียกใช้งาน และเมื่อฟังก์ชันทำงานเสร็จจะมีการคืนค่าข้อมูลกลับไปให้ฟังก์ชัน ที่เรียกใช้งาน และ **ฟังก์ชันที่มีการคืนค่ากลับและมีการรับพารามิเตอร์** (Functions Return Value with Parameters) เป็นฟังก์ชันที่สร้างขึ้นโดยมีการรับค่าข้อมูล (พารามิเตอร์) จากฟังก์ชัน ที่เรียกใช้งาน และเมื่อฟังก์ชันทำงานเสร็จ จะมีการคืนค่าข้อมูลกลับไปให้ฟังก์ชันที่เรียกใช้งาน

การใช้งานตัวแปรที่ถูกสร้างขึ้นในฟังก์ชันจะมีขอบเขตการใช้งานแตกต่างกับตัวแปรที่ถูกสร้างขึ้นนอกฟังก์ชัน ในการเขียนโปรแกรมภาษาซีตัวแปรที่มีลักษณะแตกต่างกันดังกล่าวเรียกว่าตัวแปรแบบโลคอล และ ตัวแปรแบบโกลบอล โดยตัวแปรแบบโลคอลเป็นตัวแปรที่มีการประกาศ ใช้งานภายในฟังก์ชันจึงไม่สามารถเรียกใช้งานภายนอกฟังก์ชันได้ และ ตัวแปรแบบโกลบอล เป็นตัวแปรที่มีการประกาศใช้งานตั้งแต่เริ่มต้นโปรแกรมจึงสามารถเรียกใช้งานในส่วนใดของโปรแกรมก็ได้

การที่ฟังก์ชันใดฟังก์ชันหนึ่งเรียกให้ตัวเองทำงาน จะมีชื่อเรียกเฉพาะสำหรับฟังก์ชันดังกล่าวว่ารีเคอร์ซีฟฟังก์ชัน (Recursive Function)

บทที่ 10
การจัดการข้อมูลแถวลำดับ และ ข้อความ (array and string)

10.1 ความหมายของข้อมูลแถวลำดับ หรือ อาร์เรย์

อาเรย์ คือข้อมูลแถวลำดับที่เก็บข้อมูลเป็นชุดข้อมูล โดยข้อมูลของอาเรย์จะเก็บในรูปแบบของลำดับข้อมูล ซึ่งข้อมูลนั้นจะต้องเป็นประเภทเดียวกัน ในงานประยุกต์บางอย่างอาจจะต้องมีการกระทำกับข้อมูลที่มีจำนวนมาก และเป็นข้อมูลประเภทเดียวกัน ตัวอย่างเช่น ถ้าเราต้องการคำนวณหาค่าเฉลี่ยของคะแนนการสอบของนักศึกษา จำนวน 30 คน เราจะต้องมีการกำหนดชื่อตัวแปรจำนวน 30 ชื่อ คือ score1, score2, ..., score30 เพื่อเก็บข้อมูลคะแนนสอบของนักศึกษา จำนวน 30 คน ในหน่วยความจำ ในการใช้งานจริงนั้นหากจะอ้างถึงตัวแปรทั้ง 30 ตัวซึ่งชื่อแตกต่างกันจะค่อนข้างยุ่งยาก

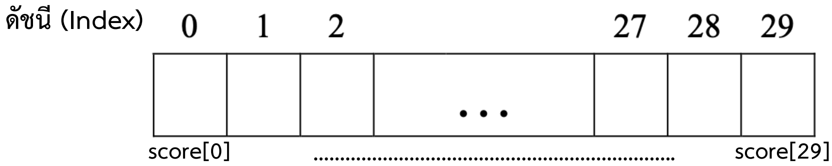
ดังนั้นการใช้ข้อมูลแถวลำดับ หรือ ตัวแปรอาร์เรย์จึงได้ถูกนำมาแก้ปัญหาลักษณะดังกล่าว โดยเราสามารถอ้างถึงชื่อตัวแปรแต่ละตัวตามด้วยตัวชี้ตำแหน่งข้อมูล หรือ ตัวบอกลำดับข้อมูล หรือ ดัชนี (index) ได้ โดยกำหนดชื่อตัวแปรตามด้วยดัชนี หรือ ตัวบอกลำดับข้อมูล โดยกำหนดอยู่ภายในเครื่องหมาย [ ] เช่น score[0], score[1] ..., score[29] ดังนี้

โดย	score[0]	**แทนตัวแปรสำหรับเก็บคะแนนนักศึกษาคนที่ 1**
	score[1]	**แทนตัวแปรสำหรับเก็บคะแนนนักศึกษาคนที่ 2**
	....	
	score[29]	**แทนตัวแปรสำหรับเก็บคะแนนนักศึกษาคนที่ 30**

หมายเหตุ *ในภาษา C ตัวบอกลำดับ หรือ ดัชนี หรือ อินเด็กซ์ของตัวแปรอาร์เรย์จะเริ่มจาก 0 ซึ่งโครงสร้างข้อมูลแบบนี้ในภาษา C เราจะเรียกว่าข้อมูลแถวลำดับ*

10.2 ข้อมูลแถวลำดับ 1 มิติ (1D array)

ข้อมูลแถวลำดับ 1 มิติ หรือ อาร์เรย์ 1 มิติ คือกลุ่มของข้อมูลที่มีจำนวนช่องข้อมูลหลายช่อง และมีชนิดของข้อมูลเป็นแบบเดียวกันทั้งหมด มีจำนวนแถว (row) ได้เพียง 1 แถว มีจำนวนคอลัมน์ (column) ได้หลายคอลัมน์ โดยจะถูกกำหนดเนื้อที่ในหน่วยความจำให้ต่อเนื่องกันไป เช่น กำหนด score[30] แสดงได้ดังรูปที่ 10.1



รูปที่ 10.1 ตัวอย่างอาร์เรย์ 1 มิติ

10.2.1 การประกาศอาร์เรย์ 1 มิติ

การประกาศอาร์เรย์ 1 มิติ จะต้องมีการกำหนดจำนวนสมาชิกในเครื่องหมาย [ ] มีรูปแบบดังนี้

```
datatype    arr_name[size]  หรือ
datatype    arr_name[size] =  {value_1, value_2, ..., value_n};
```

โดย	datatype	เป็นชนิดของตัวแปร เช่น int, float, char
	arr_name[size]	เป็นการกำหนดชื่ออาร์เรย์โดยระบุจำนวนสมาชิกในอาร์เรย์ภายในเครื่องหมาย [ ] หรือหากไม่ระบุให้ปล่อยไว้เป็นค่าว่าง
	value_1, value_2, ..., value_n	เป็นการกำหนดค่าเริ่มต้นให้กับสมาชิกลำดับที่ 1, 2, ..., n

ตัวอย่างที่ 10.1 กำหนด int x[8] และ กำหนดค่าเริ่มต้นในตัวแปรแถวลำดับ x ดังรูปที่ 10.2 พร้อมทั้งแสดงตัวอย่างคำสั่งในตารางที่ 10.1

ดัชนี (Index)	x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
ข้อมูล (data)	2	4	10	12	20	14	18	8

รูปที่ 10.2 ตัวอย่างข้อมูลอาร์เรย์ 1 มิติ

ตารางที่ 10.1 ตัวอย่างคำสั่ง

คำสั่ง	ความหมาย
printf(“%d”, x[0]);	แสดงค่าของ x[0] บนจอภาพ คือ 2
x[3] = 25;	นำค่า 25 ไปไว้ในตัวแปรอาร์เรย์ x[3]
sum = x[1] + x[6];	หาผลรวมของ x[1] กับ x[6] แล้วเก็บในตัวแปร sum
sum += x[2];	หาผลรวมของ x[2] กับ sum แล้วเก็บในตัวแปร sum

x[4] += 1;	หาผลรวมของ x[4] กับ 1 แล้วเก็บในตัวแปร x[4]
x[2] = x[0] + x[7]	หาผลรวมของ x[0] กับ x[7] แล้วเก็บในตัวแปร x[2]

เมื่อมีการกระทำการ (execute) คำสั่งทั้งหมดจะได้ผลลัพธ์ ดังรูปที่ 10.3

ดัชนี (Index)	x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
ข้อมูล (data)	2	4	10	25	21	14	18	8

รูปที่ 10.3 ผลลัพธ์ของคำสั่ง

ตัวอย่างที่ 10.2 กำหนด int arr[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}; หรื int arr[ ] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}; และทำการแสดงข้อมูลในตัวแปรอาร์เรย์ออกหน้าจอ เขียนเป็นโปรแกรมได้ดังนี้

```
1  #include <stdio.h>
2  int main() {
3      int arr[]={1,2,3,4,5,6,7,8,9,10};
4      printf("arr1 = %d\n",arr[0]);
5      printf("arr1 = %d\n",arr[1]);
6      printf("arr1 = %d\n",arr[2]);
7      printf("arr1 = %d\n",arr[3]);
8      printf("arr1 = %d\n",arr[4]);
9      printf("arr1 = %d\n",arr[5]);
10     printf("arr1 = %d\n",arr[6]);
11     printf("arr1 = %d\n",arr[7]);
12     printf("arr1 = %d\n",arr[8]);
13     printf("arr1 = %d\n",arr[9]);
14 }
```

ผลการทำงานโปรแกรม

```
arr1 = 1
arr1 = 2
arr1 = 3
arr1 = 4
arr1 = 5
arr1 = 6
arr1 = 7
arr1 = 8
arr1 = 9
arr1 = 10
```

จากตัวอย่างโปรแกรมที่ 10.2 สามารถอธิบายการทำงานดังนี้

บรรทัดที่ 3 กำหนดตัวแปรอาร์เรย์ชื่อ arr สำหรับเก็บจำนวนเต็ม (integer) โดยไม่ระบุจำนวนสมาชิกให้กับตัวแปรชื่อ arr เก็บข้อมูลชนิดจำนวนเต็ม (integer) จำนวน 10 ตัวโดยใส่ค่าเริ่มต้นให้กับอาร์เรย์ คือ 0, 1, ..., 10

บรรทัดที่ 4-10 พิมพ์ข้อมูลของอาร์เรย์ตั้งแต่ arr[0] – arr[9] แสดงผลบนหน้าจอ

ตัวอย่างที่ 10.3 กำหนด char color[3] = {'T', 'H', 'R', 'E', 'E'}; และแสดงข้อมูลออกหน้าจอ เขียนเป็นโปรแกรมได้ดังนี้

```c
1 #include <stdio.h>
2 int main() {
3     char color[5]={'T','H','R','E','E'};
4     printf("color1 = %c\n",color[0]);
5     printf("color2 = %c\n",color[1]);
6     printf("color3 = %c\n",color[2]);
7     printf("color4 = %c\n",color[3]);
8     printf("color5 = %c\n",color[4]);
9 }
```

ผลการทำงานโปรแกรม

```
color1 = T
color2 = H
color3 = R
color4 = E
color5 = E
```

จากตัวอย่างโปรแกรมที่ 10.3 สามารถอธิบายการทำงานดังนี้

บรรทัดที่ 3 กำหนดตัวแปรอาร์เรย์ชื่อ color สำหรับเก็บตัวอักษร (character) จำนวนสมาชิกเท่ากับ 5 และใส่ข้อมูลเริ่มต้นเป็นตัวอักษร 5 ตัว

บรรทัดที่ 4-8 พิมพ์ข้อมูลของอาร์เรย์ตั้งแต่ color[0] – color[4] เพื่อแสดงผลบนหน้าจอ

ตัวอย่างที่ 10.4 กำหนดค่าเริ่มต้นให้กับตัวแปรแถวลำดับแบบระบุที่ละดัชนี (index) และใช้คำสั่ง for เพื่อพิมพ์ค่าตัวแปรแถวลำดับแต่ละตัวออกสู่จอภาพ

```c
#include <stdio.h>
int main() {
    int num_array[6];
    int index;
    num_array[0] = 0;   num_array[1]= 2;
    num_array[2] = 4;   num_array[3]= 8;
    num_array[4] = 16;  num_array[5]= 24;
    for(index=0;index<=5;index++){
        printf("num_array[%d] = %d\n", index, num_array[index]);
    }
}
```

ผลการทำงานโปรแกรม

```
num_array[0] = 0
num_array[1] = 2
num_array[2] = 4
num_array[3] = 8
num_array[4] = 16
num_array[5] = 24
```

จากตัวอย่างโปรแกรมที่ 10.4 สามารถอธิบายการทำงานดังนี้

บรรทัดที่ 3 กำหนดตัวแปรอาร์เรย์ชื่อ num_array สำหรับเก็บเลขจำนวนเต็ม (integer) จำนวนสมาชิกเท่ากับ 6 โดยยังไม่ใส่ข้อมูลเริ่มต้น

บรรทัดที่ 4 กำหนดตัวแปรชื่อ index เป็นเลขจำนวนเต็ม ใช้สำหรับเป็น index ชี้ตำแหน่งข้อมูลในอาร์เรย์

บรรทัดที่ 5-7 ใส่ค่าให้กับอาร์เรย์แต่ละช่อง โดยการระบุเลขดัชนีในเครื่องหมาย [ ] ตั้งแต่ 0-5

บรรทัดที่ 8-10 ใช้คำสั่ง for โดยกำหนด index เริ่มต้นที่ 0 และสิ้นสุด index <= 5 ในการวนรอบเพื่อแสดงตำแหน่งของข้อมูล และค่าข้อมูลตัวเลขจำนวนเต็มพิมพ์ออกหน้าจอ

ตัวอย่างที่ 10.5 กำหนดค่าเริ่มต้นให้กับตัวแปรอาร์เรย์ 5 จำนวนแรก ส่วน 5 จำนวนหลัง ได้จากการคำนวณและพิมพ์ค่าทุกตัวออกสู่จอภาพ

```c
#include <stdio.h>
int main() {
    int values[10] = {0, 1, 4, 9, 16};
    int i;
    for(i=5; i<10; i++){
        values[i] = i * i;
    }
    for(i=0; i<10; i++){
        printf("values[%d] = %d\n", i, values[i]);
    }
}
```

ผลการทำงานโปรแกรม

```
values[0] = 0
values[1] = 1
values[2] = 4
values[3] = 9
values[4] = 16
values[5] = 25
values[6] = 36
values[7] = 49
values[8] = 64
values[9] = 81
```

จากตัวอย่างโปรแกรมที่ 10.5 สามารถอธิบายการทำงานดังนี้

บรรทัดที่ 3 กำหนดตัวแปรอาร์เรย์ชื่อ values สำหรับเก็บเลขจำนวนเต็ม (integer) จำนวน 10 จำนวน โดยใส่ข้อมูลเริ่มต้นเฉพาะ 5 ข้อมูลแรก มีค่า 0, 1, 4, 9, 16

บรรทัดที่ 4 กำหนดตัวแปรชื่อ i เป็นเลขจำนวนเต็ม ใช้สำหรับเป็น index ชี้ตำแหน่งข้อมูลในอาร์เรย์

บรรทัดที่ 5-7 ใช้คำสั่ง for วนรอบใส่ข้อมูลในอาร์เรย์ value โดยกำหนดตัวแปร i เริ่มต้นที่ 5 และสิ้นสุดที่ i < 10 ในการวนรอบเพื่อใส่ข้อมูลของผลคูณของค่า i

บรรทัดที่ 8-10 ใช้คำสั่ง for โดยกำหนด i เริ่มต้นที่ 0 และสิ้นสุด i < 10 ในการวนรอบเพื่อแสดงตำแหน่งของข้อมูล และค่าข้อมูลตัวเลขจำนวนเต็มของตัวแปร values และพิมพ์ออกหน้าจอ

ตัวอย่างที่ 10.6 โปรแกรมหาผลรวมของค่าในตัวแปรอาร์เรย์ทุกตัว

```c
#include <stdio.h>
#define SIZE 12
int main() {
    int arr[SIZE] = {2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24};
    int i; /*index*/
    int total = 0; /*sum of array*/

    /* sum contents of array arr*/
    for(i=0; i<SIZE; i++){
        total += arr[i];
    }
    printf("Total of array element values is %d\n",total);
}
```

ผลการทำงานโปรแกรม

```
Total of array element values is 156
```

จากตัวอย่างโปรแกรมที่ 10.6 สามารถอธิบายการทำงานดังนี้

บรรทัดที่ 2 กำหนดตัวแปรคงที่ชื่อ SIZE มีค่า 12

บรรทัดที่ 4 กำหนดตัวแปรอาร์เรย์ชื่อ arr สำหรับเก็บเลขจำนวนเต็ม (integer) จำนวนเท่ากับตัวแปร SIZE (12 จำนวน) โดยใส่ข้อมูลเริ่มต้น คือ2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24

บรรทัดที่ 5 กำหนดตัวแปรชื่อ i เป็นเลขจำนวนเต็ม ใช้สำหรับเป็น index ชี้ตำแหน่งข้อมูลในอาร์เรย์

บรรทัดที่ 6 กำหนดตัวแปรชื่อ total เป็นเลขจำนวนเต็ม มีค่าเริ่มต้นเป็น 0 ใช้สำหรับเป็นตัวแปรในการเก็บผลรวมตัวเลขในอาร์เรย์ arr ทั้งหมด

บรรทัดที่ 9-10 ใช้คำสั่ง for วนรอบใส่ข้อมูลในอาร์เรย์ arr โดยกำหนดตัวแปร i เริ่มต้นที่ 0 และสิ้นสุดที่ i < SIZE ในการวนรอบเพื่อใส่ข้อมูลของผลรวมของตัวเลขในอาร์เรย์ arr แล้วเก็บผลรวมในตัวแปร total

บรรทัดที่ 12 แสดงข้อมูลในตัวแปร total ออกหน้าจอ

ตัวอย่างที่ 10.7 โปรแกรมสำหรับรับค่าข้อมูลทางคีย์บอร์ด และเก็บค่าลงในอาร์เรย์ จากนั้นแสดงผลข้อมูลออกทางหน้าจอ

```c
#include <stdio.h>
int main() {
    float arr[5];
    int i;
    for(i=0;i<5;i++){
        printf("Enter value[%d] : ",i);
        scanf("%f",&arr[i]);
    }
    printf("\n");
    for(i=0;i<5;i++){
        printf("Value[%d] = %f\n",i,arr[i]);
    }
}
```

ผลการทำงานโปรแกรม

```
Enter value[0] : 2.2
Enter value[1] : 3.3
Enter value[2] : 4.4
Enter value[3] : 5.5
Enter value[4] : 6.6
Value[0] = 2.200000
Value[1] = 3.300000
Value[2] = 4.400000
Value[3] = 5.500000
Value[4] = 6.600000
```

จากตัวอย่างโปรแกรมที่ 10.7 สามารถอธิบายการทำงานดังนี้

บรรทัดที่ 3 กำหนดตัวแปรอาร์เรย์ชื่อ arr สำหรับเก็บเลขจำนวนจริง (float) จำนวนสมาชิกเท่ากับ 5 โดยยังไม่ใส่ค่าเริ่มต้นในอาร์เรย์ arr

บรรทัดที่ 4 กำหนดตัวแปรชื่อ i เป็นเลขจำนวนเต็ม ใช้สำหรับเป็น index ชี้ตำแหน่งข้อมูลในอาร์เรย์

บรรทัดที่ 5-8 ใช้คำสั่ง for วนรอบเพื่อแสดงข้อความและรับค่าทางคีย์บอร์ดเข้าไปในอาร์เรย์ arr โดยกำหนดตัวแปร i เริ่มต้นที่ 0 และสิ้นสุดที่ i < 5

บรรทัดที่ 10-12 ใช้คำสั่ง for วนรอบ แสดงข้อมูลในตัวแปรอาร์เรย์ arr ออกหน้าจอ

ตัวอย่างที่ 10.8 โปรแกรมสำหรับรับค่าข้อมูลจำนวนสมาชิกของตัวแปรอาร์เรย์ และข้อมูลของตัวแปรอาร์เรย์แต่ละตัวทางคีย์บอร์ด เก็บค่าลงในอาร์เรย์ จากนั้นแสดงผลข้อมูลออกทางหน้าจอ

```
1  #include <stdio.h>
2  int main() {
3      float arr[5];
4      int i, n;
5      printf("Enter amount of array : ");
6      scanf("%d",&n);
7      for(i=0;i<n;i++){
8          printf("Enter value[%d] : ",i);
9          scanf("%f",&arr[i]);
10     }
11     printf("\n");
12     for(i=0;i<n;i++){
13         printf("Value[%d] = %f\n",i,arr[i]);
14     }
15 }
```

ผลการทำงานโปรแกรม

```
Enter amount of array : 3
Enter value[0] : 2.3
Enter value[1] : 2.5
Enter value[2] : 2.7
Value[0] = 2.300000
Value[1] = 2.500000
Value[2] = 2.700000
```

จากตัวอย่างโปรแกรมที่ 10.8 สามารถอธิบายการทำงานดังนี้

บรรทัดที่ 3 กำหนดตัวแปรอาร์เรย์ชื่อ arr สำหรับเก็บเลขจำนวนจริง (float) จำนวนสมาชิกเท่ากับ 5 โดยยังไม่ใส่ค่าเริ่มต้นในอาร์เรย์ arr

บรรทัดที่ 4 กำหนดตัวแปรชื่อ i เป็นเลขจำนวนเต็ม ใช้สำหรับเป็น index ชี้ตำแหน่งข้อมูลในอาร์เรย์ และกำหนดตัวแปร n เป็นเลขจำนวนเต็ม ใช้สำหรับรับค่าจำนวนสมาชิกของอาร์เรย์

บรรทัดที่ 5 ข้อความแสดงบนหน้าจอเพื่อเตรียมรับค่าจำนวนสมาชิกอาร์เรย์ทางคีย์บอร์ด

บรรทัดที่ 6 รับค่าทางคีย์บอร์ดเป็นจำนวนเต็ม เก็บไว้ในตัวแปร n

บรรทัดที่ 7-10 ใช้คำสั่ง for วนรอบเพื่อแสดงข้อความและรับค่าทางคีย์บอร์ดเข้าไปในอาร์เรย์ arr โดยกำหนดตัวแปร i เริ่มต้นที่ 0 และสิ้นสุดที่ i < n

บรรทัดที่ 12-14 ใช้คำสั่ง for วนรอบเพื่อแสดงข้อมูลในตัวแปรอาร์เรย์ arr ออกหน้าจอ

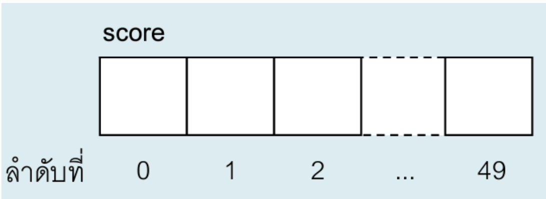
10.3 ข้อมูลแถวลำดับ 2 มิติ (2D array)

การใช้ข้อมูลแถวลำดับ 2 มิติ หรือเรียกง่าย ๆ ว่าอาร์เรย์ 2 มิติ สามารถนำมาแก้ปัญหาอาร์เรย์ 1 มิติได้ ซึ่งความสามารถของ อาเรย์ 1 มิติ ที่กล่าวไปแล้วข้างต้น สามารถเก็บข้อมูลหลาย ๆ ตัวภายในตัวแปรตัวเดียวได้ เช่นต้องการเก็บคะแนนในการสอบของนักเรียน 50 คน สามารถเก็บไว้ในตัวแปรอาร์เรย์ดังนี้

```
datatype    arr_name[size]  หรือ
datatype    arr_name[size] =  {value_1, value_2, ..., value_n};
```

```
int score[50]; หรือ int score[50] = {68, 68, ..., 95}
```

จากการประกาศตัวแปรข้างต้นจะได้โครงสร้างข้อมูลดังรูปที่ 10.4

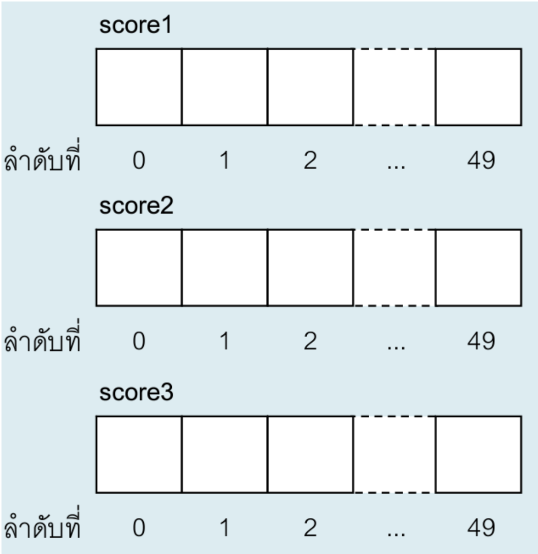


รูปที่ 10.4 ตัวแปรอาร์เรย์ 1 มิติชื่อ score ที่ขนาดสมาชิกเท่ากับ 50 ช่องข้อมูล

ซึ่งหากลองพิจารณาดูว่าถ้ามีการสอบทั้งหมด 3 วิชา เราจะต้องประกาศตัวแปรอาร์เรย์ 1 มิติ ถึง 3 ตัวแปร เพื่อเก็บข้อมูลคะแนนสอบของนักเรียนจำนวน 50 คน ทั้ง 3 วิชา ดังนี้

```
int score1[50]; /* คะแนนสอบวิชาที่ 1*/
int score2[50]; /* คะแนนสอบวิชาที่ 2*/
int score3[50]; /* คะแนนสอบวิชาที่ 3*/
```

จากการประกาศตัวแปรข้างต้นจะได้อาร์เรย์ 1 มิติ 3 ตัวแปรชื่อ score1[50], score2[50] และ score3[50]

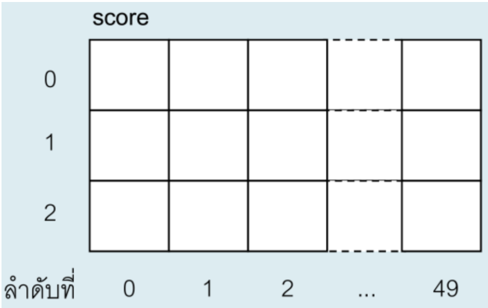


รูปที่ 10.5 ตัวแปรอาร์เรย์ 1 มิติชื่อ score1, score2 และ score3 ที่มีขนาดตัวแปรละ 50 ช่องข้อมูล

การประกาศตัวแปบแบบอาร์เรย์ที่มีชื่อตัวแปรคล้ายกันดังรูปที่ 10.5 นั้น สามารถทำให้ง่ายขึ้นได้โดยการใช้ตัวแปรแบบอาร์เรย์ 2 มิติ โดยหากต้องการตัวแปรแบบอาร์เรย์ที่มีความยาวขนาด 50 ช่อง จำนวน 3 ชุด สามารถใช้อาร์เรย์ 2 มิติโดยการประกาศดังนี้

```
int score[3][50];
```

หมายถึงกำหนดตัวแปรอาร์เรย์ชื่อ score มีแถว (row) 3 แถว แทนรายวิชา 3 วิชา และมีคอลัมน์ 50 คอลัมน์ แทนคะแนนในแต่ละวิชา โดยมีโครงสร้างการจัดเก็บข้อมูลดังรูปที่ 10.6



รูปที่ 10.5 ตัวแปรอาร์เรย์ 2 มิติชื่อ score ที่มีขนาด 3 แถว x 50 คอลัมน์

10.3.1 รูปแบบทั่วไปของการประกาศตัวแปรอาร์เรย์ 2 มิติ

```
data-type     array[size1][size2];
```

โดย	data-type	เป็นชนิดของตัวแปร เช่น int, float, char
	array	เป็นชื่อของตัวแปรอาร์เรย์ 2 มิติ ที่ตั้งตามกฎเกณฑ์การตั้งชื่อ
	size1	เป็นค่าจำนวนเต็มบวก ซึ่งอาจจะเป็นค่าคงที่หรือตัวแปร เพื่อบ่งบอกจำนวนแถว (row) ของอาร์เรย์ 2 มิติ
	size2	เป็นค่าจำนวนเต็มบวก ซึ่งอาจจะเป็นค่าคงที่หรือตัวแปร เพื่อบ่งบอกจำนวนคอลัมน์ (column) ของอาร์เรย์ 2 มิติ

ตัวอย่างที่ 10.9 กำหนด int score[3][5]; ซึ่งหมายถึงการประกาศเพื่อบอกให้คอมไพเลอร์เตรียมเนื้อที่ให้ตัวแปรอาร์เรย์ 2 มิติ ชื่อ score ให้มีเนื้อที่ต่อเนื่องกันไปจำนวน 3 rows X 5 columns = 15 ตัว โดยแต่ละตัวจะมีชนิดข้อมูลเป็น int แสดงดังรูปที่ 10.6

	column 0	column 1	column 2	column 3	column 4
row 0	score[0][0]	score[0][1]	score[0][2]	score[0][3]	score[0][4]
row 1	score[1][0]	score[1][1]	score[1][2]	score[1][3]	score[1][4]
row 2	score[2][0]	score[2][1]	score[2][2]	score[2][3]	score[2][4]

รูปที่ 10.6 ตัวแปรอาร์เรย์ 2 มิติชื่อ score ที่มีขนาด 3 แถว x 5 คอลัมน์

ตัวอย่างที่ 10.10 โปรแกรมแสดงการกำหนดค่าเริ่มต้นให้กับตัวแปรอาร์เรย์ 2 มิติ และนำค่าออกมาแสดงบนจอภาพ

```c
1  #include <stdio.h>
2  int main() {
3      int arr[3][4] = {{1,2,3,4},
4                       {5,6,7,8},
5                       {9,10,11,12}};
6      int row,col;
7      for(row=0; row<3; row++){
8          for(col=0; col<4; col++){
9              printf("%d\t",arr[row][col]);
10         }
11     printf("\n");
12     }
13 }
```

ผลการทำงานโปรแกรม

```
1	2	3	4
5	6	7	8
9	10	11	12
```

จากตัวอย่างโปรแกรมที่ 10.10 สามารถอธิบายการทำงานดังนี้

บรรทัดที่ 3-5 กำหนดตัวแปรอาร์เรย์ 2 มิติ ชื่อ arr โดยมีขนาด 3 แถว (row) 4 คอลัมน์ (column) และกำหนดค่าเริ่มต้นให้กับอาร์เรย์ 2 มิติ arr

บรรทัดที่ 6 กำหนดตัวแปรสำหรับเก็บค่าเลขจำนวนเต็ม 2 ตัว ชื่อ row และ col เพื่อเป็นตัวชี้ (index) ตำแหน่งของแถวและคอลัมน์ในอาร์เรย์ 2 มิติ

บรรทัดที่ 7-10 คำสั่ง for ใช้ในการวนรอบอาร์เรย์ 2 มิติ เพื่อดึงข้อมูลออกมาแสดงผลบนหน้าจอ โดย for ชั้นที่ 1 ในบรรทัดที่ 7 เป็นการสั่งให้วนรอบตามแนวแถว (row) และ for ชั้นที่ 2 ในบรรทัดที่ 8 เป็นการสั่งให้วนรอบตามแนวคอลัมน์ (column) โดยกำหนดเป็น arr[row][col]

ตัวอย่างที่ 10.11 โปรแกรมสำหรับรับค่าข้อมูลคะแนนของนักศึกษา 2 คน 3 รายวิชาทางคีย์บอร์ด แล้วใส่ในตัวแปรอาร์เรย์สองมิติ ขนาด 2X3 จากนั้นแสดงผลลัพธ์ออกหน้าจอ

```
1   #include <stdio.h>
2   int main() {
3       int score[2][3];
4       int row, col;
5       for(row=0; row<2; row++){
6           for(col=0; col<3; col++){
7               printf("Enter score student[%d] subject[%d] : ",row,col);
8               scanf("%d",&score[row][col]);
9           }
10      }
11      for(row=0; row<2; row++){
12          for(col=0; col<3; col++){
13              printf("student[%d] subject[%d] score : %d\n",row,col
                    ,score[row][col]);
14          }
15          printf("\n");
16      }
17  }
```

ผลการทำงานโปรแกรม

```
Enter score student[0] subject[0] : 80
Enter score student[0] subject[1] : 89
Enter score student[0] subject[2] : 90
Enter score student[1] subject[0] : 75
Enter score student[1] subject[1] : 78
Enter score student[1] subject[2] : 92
student[0] subject[0] score : 80
student[0] subject[1] score : 89
student[0] subject[2] score : 90

student[1] subject[0] score : 75
student[1] subject[1] score : 78
student[1] subject[2] score : 92
```

จากตัวอย่างโปรแกรมที่ 10.11 สามารถอธิบายการทำงานดังนี้

บรรทัดที่ 3 กำหนดตัวแปรอาร์เรย์ 2 มิติ ชื่อ score โดยมีจำนวนแถว (row) 2 และจำนวนคอลัมน์ (column) 3 และยังไม่ใส่ค่าเริ่มต้นให้อาร์เรย์

บรรทัดที่ 4 กำหนดตัวแปรชื่อ row และ col เป็นชนิดตัวเลขจำนวนเต็ม สำหรับใช้เป็นตัวชี้ข้อมูลของตัวแปรอาร์เรย์ score

บรรทัดที่ 5 ใช้คำสั่ง for สำหรับวนรอบข้อมูลอาร์เรย์ 2 มิติ ในแนวแถว (row) โดยมีตัวชี้ (index) เป็นตัวแปร row

บรรทัดที่ 6 ใช้คำสั่ง for สำหรับวนรอบข้อมูลอาร์เรย์ 2 มิติ ในแนวคอลัมน์ (column) โดยมีตัวชี้ (index) เป็นตัวแปร col

บรรทัดที่ 7 ข้อความแสดงบนหน้าจอเพื่อเตรียมรับค่าข้อมูลทางคีย์บอร์ด พร้อมทั้งแสดงตำแหน่ง row และ col ของอาร์เรย์ score

บรรทัดที่ 8 รับข้อมูลทางคีย์บอร์ดใส่ในตัวแปรอาร์เรย์ score

บรรทัดที่ 11 ใช้คำสั่ง for ชั้นที่ 1 สำหรับวนรอบข้อมูลอาร์เรย์ 2 มิติ ในแนวแถว (row) โดยมีตัวชี้ (index) เป็นตัวแปร row

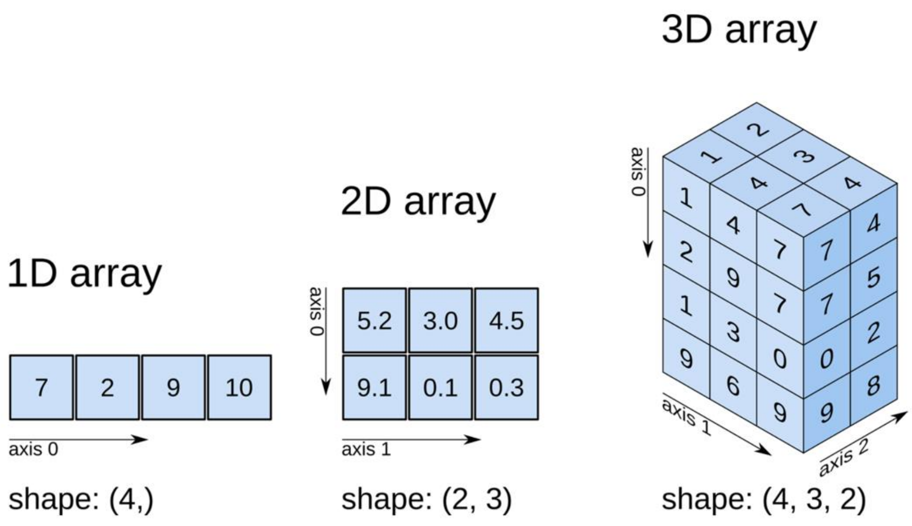
บรรทัดที่ 12 ใช้คำสั่ง for ชั้นที่ 2 สำหรับวนรอบข้อมูลอาร์เรย์ 2 มิติ ในแนวคอลัมน์ (column) โดยมีตัวชี้ (index) เป็นตัวแปร col

บรรทัดที่ 13 แสดงข้อมูลในตัวแปร score โดยดัชนี row เป็นนักศึกษา และดัชนี col เป็นรายวิชา

10.4 ข้อมูลแถวลำดับ 3 มิติ (3D array)

เนื่องจากตัวแปรอาร์เรย์สามารถสร้างได้หลายมิติ คือ 3, 4, ... N มิติ เมื่อเราเข้าใจตัวแปรอาร์เรย์ 1 มิติ และ 2 มิติ แล้ว ก็จะสามารถเพิ่มมิติเข้าไปได้ โดยปกติแล้ว มากกว่า 2 มิติ จะไม่ค่อยได้ใช้ ยกเว้นการเขียนโปรแกรมจัดการข้อมูลที่มีความซับซ้อนในทางวิศวกรรมคอมพิวเตอร์ ในบทนี้จะกล่าวถึงแค่เพียงอาร์เรย์ 3 มิติ เพื่อให้เข้าใจการจัดเก็บข้อมูล และการจัดการข้อมูลอาร์เรย์ 3 มิติ

จากรูปที่ 10.7 แสดงการจำลองโครงสร้างข้อมูลของอาร์เรย์ 1, 2 และ 3 มิติ โดยอาร์เรย์ 3 มิตินั้นจะมีรูปแบบในการเก็บข้อมูลเป็นลักษณะของสี่เหลี่ยมลูกบาศก์ (cubic) โดยมีมิติความลึก (dept) ของอาร์เรย์เข้ามาเพิ่มเติมจากอาร์เรย์ 2 มิติ โดยรูปอาร์เรย์สามมิตินั้นจะต้องมีการระบุตำแหน่งของอาร์เรย์ 3 แกน คือ [แถว][คอลัมน์][ความลึก] หรือ [row][column][dept]



รูปที่ 10.7 จำลองโครงสร้างการเก็บข้อมูลของอาร์เรย์ 1, 2 และ 3 มิติ

(ที่มา : https://i.stack.imgur.com/Tbe9W.png)

10.4.1 รูปแบบทั่วไปของการประกาศตัวแปรอาร์เรย์ 3 มิติ

```
data-type      array[size1][size2][size3];
```

โดย	data-type	เป็นชนิดของตัวแปร เช่น int, float, char
	array	เป็นชื่อของตัวแปรอาร์เรย์ 3 มิติ ที่ตั้งตามกฎเกณฑ์การตั้งชื่อ
	size1	เป็นค่าจำนวนเต็มบวก ซึ่งอาจจะเป็นค่าคงที่หรือตัวแปร เพื่อบ่งบอกจำนวนแถว (row) ของอาร์เรย์ 3 มิติ
	size2	เป็นค่าจำนวนเต็มบวก ซึ่งอาจจะเป็นค่าคงที่หรือตัวแปร เพื่อบ่งบอกจำนวนคอลัมน์ (column) ของอาร์เรย์ 3 มิติ
	Size3	เป็นค่าจำนวนเต็มบวก ซึ่งอาจจะเป็นค่าคงที่หรือตัวแปร เพื่อบ่งบอกจำนวนความลึก (dept) ของอาร์เรย์ 3 มิติ

ตัวอย่างที่ 10.12 โปรแกรมสร้างตัวแปรอาร์เรย์ 3 มิติ ชื่อ test[2][3][2] เป็นตัวแปรชนิดจำนวนเต็ม (int) และรับค่าข้อมูลเข้ามาทางคีย์บอร์ด จากนั้นแสดงข้อมูลออกทางหน้าจอ

```c
#include <stdio.h>
int main()
{
  int test[2][3][2];
  printf("Enter 12 values: \n");
  for (int i = 0; i < 2; i++){
    for (int j = 0; j < 3; j++){
      for (int k = 0; k < 2; k++){
        scanf("%d", &test[i][j][k]);
      }
    }
  }
  printf("\nDisplaying values:\n");
  for (int i = 0; i < 2; i++){
    for (int j = 0; j < 3; j++){
      for (int k = 0; k < 2; k++){
        printf("test[%d][%d][%d] = %d\n", i,j,k,test[i][j][k]);
      }
    }
  }
}
```

ผลการทำงานโปรแกรม	
``Enter 12 values: 1 2 3 4 5 6 7 8 9 10 11 12```	```Displaying values: test[0][0][0] = 1 test[0][0][1] = 2 test[0][1][0] = 3 test[0][1][1] = 4 test[0][2][0] = 5 test[0][2][1] = 6 test[1][0][0] = 7 test[1][0][1] = 8 test[1][1][0] = 9 test[1][1][1] = 10 test[1][2][0] = 11 test[1][2][1] = 12```

จากตัวอย่างโปรแกรมที่ 10.12 สามารถอธิบายการทำงานดังนี้

บรรทัดที่ 4 กำหนดตัวแปรอาร์เรย์ 3 มิติ ชื่อ test โดยมีจำนวนแถว (row) 2 และจำนวนคอลัมน์ (column) 3 และความลึก (dept) 2และยังไม่ใส่ค่าเริ่มต้นให้อาร์เรย์

บรรทัดที่ 5 ข้อความแสดงหน้าจอเพื่อเตรียมรับค่าทางคีย์บอร์ด

บรรทัดที่ 6 คำสั่ง for ชั้นที่ 1 วนรอบในแนวแถว (row)

บรรทัดที่ 7 คำสั่ง for ชั้นที่ 2 วนรอบในแนวคอลัมน์ (column)

บรรทัดที่ 8 คำสั่ง for ชั้นที่ 3 วนรอบในแนวลึก (dept)

บรรทัดที่ 9 รับค่าเลขจำนวนเต็มทางคีย์บอร์ดเข้ามาใส่ในตัวแปรอาร์เรย์ 3 มิติ ชื่อ test

บรรทัดที่ 14 คำสั่ง for ชั้นที่ 1 วนรอบในแนวแถว (row)

บรรทัดที่ 15 คำสั่ง for ชั้นที่ 2 วนรอบในแนวคอลัมน์ (column)

บรรทัดที่ 16 คำสั่ง for ชั้นที่ 3 วนรอบในแนวลึก (dept)

บรรทัดที่ 17 แสดงข้อมูลในตัวแปรอาร์เรย์ 3 มิติชื่อ test ออกทางจอภาพ พร้อมทั้งแสดงตำแหน่งการจัดเก็บข้อมูลในตำแหน่ง row, column และ dept ออกมาด้วย

10.5 การใช้ตัวแปรอาร์เรย์ผ่านค่าให้ฟังก์ชัน

ในภาษา C การผ่านค่าตัวแปรอาร์เรย์ไปยังฟังก์ชัน เป็นการผ่านค่าโดยการอ้างอิง หรือ call by reference

ตัวอย่างที่ 10.13 โปรแกรมแสดงการผ่านค่าตัวแปรอาร์เรย์ 1 มิติชื่อ array ไปให้ฟังก์ชัน

```
1  #include <stdio.h>
2  void function1(int arr[]);
3  int main() {
4      int array[3];
5      array[0] = 100;
6      array[1] = 200;
7      array[2] = 300;
8      function1(array);
9  }
10 void function1(int arr[]){
11     printf("contents of arry[0] => %d\n",arr[0]);
12     printf("contents of arry[1] => %d\n",arr[1]);
13     printf("contents of arry[2] => %d\n",arr[2]);
14 }
```

ผลการทำงานโปรแกรม

```
contents of arry[0] => 100
contents of arry[1] => 200
contents of arry[2] => 300
```

จากตัวอย่างโปรแกรมที่ 10.13 สามารถอธิบายการทำงานดังนี้

บรรทัดที่ 2 กำหนดโปรโตไทป์ (prototype) ของ function1

บรรทัดที่ 4 กำหนดตัวแปรอาร์เรย์ชื่อ array สำหรับเก็บเลขจำนวนเต็ม (int) จำนวน 3 ช่องข้อมูล โดยยังไม่ใส่ค่าเริ่มต้นในอาร์เรย์ array

บรรทัดที่ 5-7 ใส่เลขจำนวนเต็มให้ตัวแปร array ทีละตัว

บรรทัดที่ 8 เรียนใช้ funtion1 และส่งค่าตัวแปร array ไปยัง funtion1 การส่งค่าตัวแปรไปยังฟังก์ชันไม่ต้องใส่เครื่องหมาย [ ]

บรรทัดที่ 10 กำหนด function1 ให้มีพารามิเตอร์ชื่อ arr[ ] เพื่อรับข้อมูลจากจุดที่เรียกใช้ function1 ในฟังก์ชัน main การระบุพารามิเตอร์ arr[ ] จำเป็นต้องใส่เครื่องหมาย [ ] ด้วย

บรรทัดที่ 11–13 แสดงผลข้อมูลที่อยู่ในตัวแปรอาร์เรย์ arr ทีละตัวออกทางจอภาพ

ตัวอย่างที่ 10.14 โปรแกรมสำหรับให้ผู้ใช้ป้อนข้อมูลจำนวน 5 ตัว เพื่อเก็บลงในตัวแปรอาร์เรย์ 1 มิติชื่อ arr หลังจากนั้นเรียกใช้ฟังก์ชันเพื่อพิมพ์ค่าย้อนกลับ

```
1  #include <stdio.h>
2  #define maxnum 5
3  void backward(int arr[]);
4  int main() {
5      int number[maxnum];
6      int index;
7      printf("Plese enter %d number\n",maxnum);
8      for(index=0; index<maxnum; index++){
9          printf("Number[%d] = ",index);
10         scanf("%d",&number[index]);
11     }
12     printf("Thank you ...\n");
13     backward(number);
14 }
15 void backward(int arr[]){
16     int index;
17     printf("\nHere are you backward numbers\n");
18     for(index=maxnum-1; index>=0; index--){
19         printf("number[%d] = %d\n", index, arr[index]);
20     }
21 }
```

ผลการทำงานโปรแกรม

```
Plese enter 5 number
Number[0] = 10
Number[1] = 20
Number[2] = 30
Number[3] = 40
Number[4] = 50
Thank you ...

Here are you backward numbers
number[4] = 50
number[3] = 40
number[2] = 30
number[1] = 20
number[0] = 10
```

จากตัวอย่างโปรแกรมที่ 10.14 สามารถอธิบายการทำงานดังนี้

บรรทัดที่ 2 กำหนดค่าคงที่ให้ตัวแปร maxnum มีค่าเท่ากับ 5

บรรทัดที่ 3 กำหนดโปรโตไทป์ (prototype) ของฟังก์ชัน backward

บรรทัดที่ 5 กำหนดตัวแปรอาร์เรย์ชื่อ number สำหรับเก็บเลขจำนวนเต็ม (int) จำนวนเท่ากับตัวแปร maxnum หรือ 5 เซลล์ โดยยังไม่ใส่ค่าเริ่มต้นในอาร์เรย์ number

บรรทัดที่ 6 กำหนดตัวแปรชื่อ index เป็นเลขจำนวนเต็ม เพื่อใช้เป็นตัวชี้ข้อมูล (index)

บรรทัดที่ 7 ข้อความแสดงบนหน้าจอเพื่อเตรียมรับค่าข้อมูลทางคีย์บอร์ด พร้อมทั้งแสดงจำนวนค่าที่ user ต้องคีย์ ด้วยตัวแปร maxnum

บรรทัดที่ 8 คำสั่ง for สำหรับวนรอบในการรับค่า โดยค่าเริ่มต้นเป็น index=0; และสิ้นสุดที่ index < maxnum; และ index เพิ่มค่าทีละ 1

บรรทัดที่ 9 ข้อความแสดงบนหน้าจอเพื่อเตรียมรับค่าเข้าแต่ละเซลล์อาร์เรย์ทางคีย์บอร์ด พร้อมทั้งแสดงค่า index ปัจจุบันที่กำลังป้อนข้อมูล

บรรทัดที่ 10 คำสั่ง scanf รับค่าทางคีย์บอร์ดใส่ค่าลงในตัวแปร number

บรรทัดที่ 12 พิมพ์ข้อความ Thank you ออกหน้าจอ

บรรทัดที่ 13 เรียกใช้ฟังก์ชัน backward พร้อมส่งค่าตัวแปรอาร์เรย์ number ไปฟังก์ชัน backward

บรรทัดที่ 15 กำหนดฟังก์ชัน backward โดยมีการรับพารามิเตอร์ คือ int arr[ ]

บรรทัดที่ 16 กำหนดตัวแปร index เป็นชนิดเลขจำนวนเต็ม

บรรทัดที่ 17 พิมพ์ข้อความเพื่อเตรียมแสดงผลข้อมูลออกหน้าจอ

บรรทัดที่ 18 คำสั่ง for สำหรับวนรอบในการรับค่า โดยค่าเริ่มต้นเป็น index=maxnum-1; และสิ้นสุดที่ index >= 0; และ index ลดค่าลงทีละ 1

บรรทัดที่ 19 แสดงข้อมูลตัวเลขที่พิมพ์ค่าย้อนกลับออกหน้าจอ

ตัวอย่างที่ 10.15 โปรแกรมสำหรับให้ผู้ใช้ป้อนข้อมูลตัวเลขจำนวนเต็มทางคีย์บอร์ดจำนวน 5 ตัว จากนั้นเรียกใช้ฟังก์ชันเพื่อให้ค้นหาจำนวนตัวเลขที่มีค่าน้อยที่สุด

```
1  #include <stdio.h>
2  #define maxnum 5
3  int min (int array[]);
4  int main() {
5      int number[maxnum];
6      int index;
7      printf("Plese enter %d number\n",maxnum);
8      for(index=0; index<maxnum; index++){
9          printf("Number[%d] = ",index);
10         scanf("%d",&number[index]);
11     }
12     printf("Thank you ...\n");
13     printf("The mininum value is %d\n",min(number));
14 }
15 int min(int array[]){
16     int index;
17     int min_value;
18     min_value = array[0];
19     for(index=0; index<maxnum; index++){
20         if(array[index] < min_value)
21         {
22             min_value = array[index];
23         }
24     }
25     return(min_value);
26 }
```

ผลการทำงานโปรแกรม

```
Plese enter 5 number
Number[0] = 90
Number[1] = 80
Number[2] = 110
Number[3] = 230
Number[4] = 29
Thank you ...
The mininum value is 29
```

จากตัวอย่างโปรแกรมที่ 10.15 สามารถอธิบายการทำงานดังนี้

บรรทัดที่ 2 กำหนดตัวแปรเก็บค่าคงที่ชื่อ maxnum มีค่าเท่ากับ 5

บรรทัดที่ 3 กำหนดโปรโตไทป์ของฟังก์ชัน min

บรรทัดที่ 5 กำหนดตัวแปรอาร์เรย์ชื่อ number สำหรับเก็บเลขจำนวนเต็ม (char) มีจำนวนเซลล์เท่ากับตัวแปร maxnum หรือ ค่า 5

บรรทัดที่ 6 กำหนดตัวแปรชื่อ index สำหรับเป็นตัวชี้ข้อมูล

บรรทัดที่ 7 ข้อความแสดงบนหน้าจอเพื่อเตรียมรับค่าตัวเลขจากทางคีย์บอร์ด และนำไปเก็บในตัวแปร number

บรรทัดที่ 8-11 คำสั่ง for สำหรับวนรอบรับค่าตัวเลขทางคีย์บอร์ดด้วยคำสั่ง scanf( ) เข้าตัวแปร number พร้อมทั้งแสดงตัวเลขลำดับของ index ด้วย

บรรทัดที่ 12 พิมพ์ข้อความ Thank you ออกหน้าจอ

บรรทัดที่ 13 เรียกใช้ฟังก์ชัน min โดยส่งค่าตัวแปร number ไปยังฟังก์ชัน min หลังจากนั้นรับค่าที่ return กลับมาจากฟัง min มาทำการพิมพ์ออกหน้าจอ โดยใช้คำสั่ง printf( )

บรรทัดที่ 15 กำหนดฟังก์ชันชื่อ min โดยมีพารามิเตอร์เป็นจำนวนเต็มชื่อ array[ ]

บรรทัดที่ 16 กำหนดตัวแปร index เป็นชนิดเลขจำนวนเต็ม เพื่อใช้เป็นตัวชี้ข้อมูล

บรรทัดที่ 17 กำหนดตัวแปร min_value เป็นชนิดเลขจำนวนเต็ม เพื่อใช้เป็นตัวเลขที่มีค่าน้อยที่สุด

บรรทัดที่ 18 ให้ตัวแปร min_value เริ่มต้นมีค่าเท่ากับ array[0]

บรรทัดที่ 19 คำสั่ง for สำหรับวนรอบการทำงานการค่าตัวเลขที่น้อยที่สุด โดยกำหนด index=0; index > maxnum; และ เพิ่มค่า index ทีละ 1

บรรทัดที่ 20-23 เงื่อนไข if เพื่อทำการตรวจสอบว่า หากตัวแปร array ตัวปัจจุบันมีค่าน้อยกว่าตัวแปร min_value ให้ทำการสลับค่าในตัวแปร array มาเก็บไว้ในตัวแปร min_valu

บรรทัดที่ 25 ทำการส่งตัวแปร min_value กลับไปยังจุดที่เรียกใช้ฟังก์ชัน min ในฟังก์ชัน main

10.6 สตริง (string)

สตริง คือ ลักษณะของข้อมูลที่มาจากกลุ่มของตัวอักษร (character) หรือเรียกว่า array of character สตริงจะถูกนำมาใช้ในการเก็บข้อมูลประเภท ชื่อ, นามสกุล, ที่อยู่ หรือ เบอร์โทรศัพท์ ในภาษา C ไม่มีโครงสร้างของตัวแปรชนิดนี้มาให้เป็นมาตรฐาน ดังนั้นจึงต้องประยุกต์โครงสร้างของตัวแปรอาร์เรย์เพื่อนำมาเก็บข้อมูลสตริง ข้อมูลที่เป็นสตริงนั้นจะไม่สามารถนำมาคำนวณได้

10.6.1 การประกาศและกำหนดค่าตัวแปรสตริง

สำหรับการกำหนดค่าให้กับตัวแปรสตริงนั้น มีรูปแบบการกำหนดค่า เช่น กำหนดข้อความหรือสตริงภายใต้เครื่องหมาย “ ” ต่อเนื่องกันไป หรือ กำหนดตัวอักษรทีละตัวโดยต้องอยู่ภายใต้เครื่องหมายวงเล็บปีกกา

{‘’, ‘’, ...} กรณีกำหนดตัวอักษรทีละตัวตัวสุดท้ายจำเป็นจะต้องมีเครื่องหมาย \0 ตัวอย่างการกำหนดค่าตัวแปรสตริง แสดงดังนี้

```
char str[] = “RMUTI”;
char str[10] = “Korat”;
char str[] = {‘a’, ‘b’, ‘c’, ‘d’, ‘\0’};
char phoneno[] = {‘0’, ‘8’, ‘6’, ‘8’, ‘1’, ‘2’, ‘9’, ‘1’, ‘2’, ‘7’, ‘\0’};
char address[] = “RMUTI Nakornrachasima”;
```

10.6.2 การรับและแสดงผลข้อมูลสตริง

การรับข้อมูลใส่ในตัวแปรสตริง หรือ array of string จะใช้คำสั่ง **gets( )** หรือ **scanf** ก็ได้ ส่วนการแสดงผลข้อมูลอออกทางหน้าจอจะใช้ได้คำสั่ง **printf( )** โดยต้องระบุ format code สำหรับแสดงผล เป็น %s หรือการแสดงผลข้อมูลจะใช้คำสั่ง **puts( )** ได้อีกหนึ่งคำสั่ง

ตัวอย่างที่ 10.16 โปรแกรมรับและแสดงผลข้อมูลสตริง โดยใช้คำสั่ง gets( ) และ printf( )

```
1 #include <stdio.h>
2 int main() {
3     char str[50];
4     printf("Please enter string : ");
5     gets(str);
6     printf("Your string is : %s",str);
7 }
```

ผลการทำงานโปรแกรม

```
Please enter string : RMUTI Korat
Your string is : RMUTI Korat
```

จากตัวอย่างโปรแกรมที่ 10.16 สามารถอธิบายการทำงานดังนี้

บรรทัดที่ 3 กำหนดตัวแปรอาร์เรย์เก็บข้อมูลสตริงชื่อ str จำนวน 50 ตัวอักษร โดยยังไม่ใส่ค่าเริ่มต้นในอาร์เรย์ str

บรรทัดที่ 4 ข้อความแสดงบนหน้าจอเพื่อเตรียมรับค่าสตริงทางคีย์บอร์ด

บรรทัดที่ 5 คำสั่ง gets(str) เพื่อรับข้อความทางคีย์บอร์ดเก็บลงในตัวแปร str

บรรทัดที่ 6 พิมพ์ค่าจากตัวแปร str ออกหน้าจอ โดยใช้คำสั่ง printf( )

ตัวอย่างที่ 10.17 โปรแกรมรับและแสดงผลข้อมูลสตริง โดยใช้คำสั่ง gets( ) และ puts( )

```
1  #include <stdio.h>
2  int main() {
3      char str[80];
4      printf("Enter name : ");
5      gets(str);
6      printf("Your name : ");
7      puts(str);
8  }
```

ผลการทำงานโปรแกรม

```
Enter name : Computer Engineering RMUTI
Your name : Computer Engineering RMUTI
```

จากตัวอย่างโปรแกรมที่ 10.17 สามารถอธิบายการทำงานดังนี้

บรรทัดที่ 3 กำหนดตัวแปรอาร์เรย์เก็บข้อมูลสตริงชื่อ str จำนวน 80 ตัวอักษร โดยยังไม่ใส่ค่าเริ่มต้นในอาร์เรย์ str

บรรทัดที่ 4 ข้อความแสดงบนหน้าจอเพื่อเตรียมรับค่าสตริงทางคีย์บอร์ด

บรรทัดที่ 5 คำสั่ง gets(str) เพื่อรับข้อความทางคีย์บอร์ดเก็บลงในตัวแปร str

บรรทัดที่ 4 ข้อความแสดงบนหน้าจอเพื่อเตรียมแสดงค่าทางจอภาพ

บรรทัดที่ 6 พิมพ์ค่าจากตัวแปร str ออกหน้าจอ โดยใช้คำสั่ง puts(str);

ตัวอย่างที่ 10.18 โปรแกรมสำหรับรับข้อมูลจากคีย์บอร์ดเป็นสตริง จำนวน 50 ตัวอักษรเข้าไปในอาร์เรย์สตริงทีละตัว และนำค่ามาแสดงผลบนหน้าจอ

```
#include <stdio.h>
int main() {
    char str[50];
    int idx;
    printf("Please enter string : ");
    gets(str);
    for(idx=0; str[idx]; idx++){
        printf("Your string is str[%d] : %c\n",idx,str[idx]);
    }
}
```

ผลการทำงานโปรแกรม

```
Please enter string : RMUTI Korat
Your string is str[0] : R
Your string is str[1] : M
Your string is str[2] : U
Your string is str[3] : T
Your string is str[4] : I
Your string is str[5] :
Your string is str[6] : K
Your string is str[7] : o
Your string is str[8] : r
Your string is str[9] : a
Your string is str[10] : t
```

จากตัวอย่างโปรแกรมที่ 10.18 สามารถอธิบายการทำงานดังนี้

บรรทัดที่ 3 กำหนดตัวแปรอาร์เรย์เก็บข้อมูลสตริงชื่อ str จำนวน 50 ตัวอักษร โดยยังไม่ใส่ค่าเริ่มต้นในอาร์เรย์ str

บรรทัดที่ 4 กำหนดตัวแปร idx เป็นเลขจำนวนเต็มเพื่อใช้เป็นตัวชี้ข้อมูล (index)

บรรทัดที่ 5 ข้อความแสดงบนหน้าจอเพื่อเตรียมรับค่าสตริงทางคีย์บอร์ด

บรรทัดที่ 5 คำสั่ง gets(str) เพื่อรับข้อความทางคีย์บอร์ดเก็บลงในตัวแปร str

บรรทัดที่ 7 คำสั่ง for ในการวนรอบแสดงผลข้อมูลในตัวแปร str โดยกำหนดค่า idx=0 กรณีการวนรอบค่าในสตริง เทอมกลางของคำสั่ง for เขียนคำสั่งเพียงแค่ str[idx] เท่านั้น

บรรทัดที่ 8 แสดงลำดับข้อมูลและค่าในตัวแปร str ออกหน้าจอ

10.6.3 ฟังก์ชันมาตรฐานสำหรับ string

ฟังก์ชันมาตรฐานที่ใช้ทำงานกับข้อมูลสตริงมีอยู่หลายคำสั่ง และสามารถเรียกใช้ได้โดยง่ายและไม่ต้องเขียนโปรแกรมยาวหลายบรรทัด ซึ่งก่อนการใช้งานฟังก์ชันสำหรับ string นั้น จะต้องมีการกำหนด #include <string.h> ไว้ก่อนเสมอ ฟังก์ชันต่าง ๆ แสดงดังตารางที่ 10.2

ตารางที่ 10.2 แสดงฟังก์ชันมาตรฐานที่ใช้งานกับสตริง

ฟังก์ชัน	Syntax	คำอธิบาย
strlen	int strlen(const char *str)	นับจำนวนตัวอักษรในตัวแปรสตริง ตามจำนวนจริงของตัวอักษร
sizeof	int sizeof(const char *str)	นับความยาวของหน่วยความจำที่สตริงจองทั้งหมด
strcpy	char *strcpy(char *str1, char *str2)	คัดลอกข้อมูลใน str2 ไปยัง str1

ฟังก์ชัน	Syntax	คำอธิบาย
strcmp	int strcmp(const char *str1,const *str2)	เปรียบเทียบสตริงสองตัว โดยถ้ามีค่าเท่ากันจะคืนค่า 0 แต่ถ้าหากไม่เท่ากันจะแสดงเป็นค่า + เมื่อ str1>str2 และจะแสดงค่า – เมื่อ str1<str2
strcat	char *strcat(char *str1, char str2)	เชื่อมต่อสตริง str1 กับ str2
strchr	char *strchr(char *str, int ch)	ค้นหาตัวอักษรที่อยู่ในสตริง str โดยหากเจอตัวอักษรค้นหา จะแสดงสตริงตั้งแต่ตัวอักษรที่เจอจนตัวอักษรสุดท้ายในสตริง
strstr	char *strstr(char *str, char *srch_term)	ค้นหาบางส่วนของสตริงที่อยู่ในสตริง str โดยหากเจอบางส่วนของสตริงค้นหา จะแสดงสตริงตั้งแต่ส่วนของสตริงที่เจอจนตัวอักษรสุดท้ายในสตริง
strlwr	char *strlwr(char *str)	แปลงตัวอักษรในสตริงให้เป็นตัวพิมพ์เล็ก
strupr	char *strupr(char *str)	แปลงตัวอักษรในสตริงให้เป็นตัวพิมพ์ใหญ่

ตัวอย่างที่ 10.19 โปรแกรมที่ใช้ฟังก์ชันมาตรฐานตามตารางที่ 10.2 และพิมพ์ผลลัพธ์จากฟังก์ชันออกหน้าจอ

```c
#include <stdio.h>
int main() {
    char mystr1[20] = "Korat";
    char mystr2[30] = " RMUTI KORAT";
    printf("function strlen : %d\n",strlen(mystr1));
    printf("function sizeof : %d\n",sizeof(mystr1));
    printf("function strcmp : %d\n",strcmp(mystr1,mystr2));
    printf("function strcpy : %s\n",strcpy(mystr1,mystr2));
    printf("function strcmp : %d\n",strcmp(mystr1,mystr2));
    printf("function strcat : %s\n",strcat(mystr1,mystr2));
    printf("function strchr : %s\n",strchr(mystr1,'K'));
    printf("function strstr : %s\n",strstr(mystr2,"I KORAT"));
    printf("function mystr1 : %s\n", mystr1);
    printf("function mystr2 : %s\n", mystr2);
    printf("function strlwr : %s\n", strlwr(mystr1));
    printf("function strupr : %s\n", strupr(mystr2));
}
```

ผลการทำงานโปรแกรม

function strlen : 5

function sizeof : 20

function strcmp : 43

```
function strcpy :  RMUTI KORAT
function strcmp : 0
function strcat :  RMUTI KORAT RMUTI KORAT
function strchr : KORAT RMUTI KORAT
function strstr : I KORAT
function mystr1 :  RMUTI KORAT RMUTI KORAT
function mystr2 :  RMUTI KORAT
function strlwr : rmuti korat rmuti korat
function strupr : RMUTI KORAT
```

จากตัวอย่างโปรแกรมที่ 10.19 สามารถอธิบายการทำงานดังนี้

บรรทัดที่ 3 กำหนดตัวแปรอาร์เรย์ชื่อ mystr1 จำนวน 20 เซลล์ และใส่ค่าสตริง “Korat”

บรรทัดที่ 4 กำหนดตัวแปรอาร์เรย์ชื่อ mystr2 จำนวน 30 เซลล์ และใส่ค่าสตริง “ RMUTI KORAT”

บรรทัดที่ 5 แสดงจำนวนตัวอักษรในตัวแปร mystr1

บรรทัดที่ 6 แสดงจำนวนเซลล์ที่ตัวแปร mystr1 จองไว้

บรรทัดที่ 7 แสดงการเปรียบเทียบตัวอักษรระหว่าง mystr1 กับ mystr2

บรรทัดที่ 8 คัดลอกข้อมูลจาก mystr2 มาทับลงในข้อมูล mystr1 นั่นความหมายว่าข้อมูลเดิมใน mystr1 จะถูกแทนที่ด้วยข้อมูลจาก mystr2

บรรทัดที่ 9 แสดงการเปรียบเทียบตัวอักษรระหว่าง mystr1 กับ mystr2 หลังจากมีการคัดลอก

บรรทัดที่ 10 แสดงการเชื่อมต่อข้อมูลตัวแปร mystr1 กับ mystr2 ซึ่งผลของการเชื่อมต่อจะถูกเก็บลงในตัวแปร mystr1

บรรทัดที่ 11 ค้นหาตัวอักษร ‘K’ ในตัวแปร mystr1

บรรทัดที่ 12 ค้นหาข้อความ “I KORAT” ในตัวแปร mystr2

บรรทัดที่ 13-14 แสดงข้อความในตัวแปร mystr1 และ mystr2

บรรทัดที่ 15 แสดงข้อความในตัวแปร mystr1 เป็นตัวอักษรพิมพ์เล็กทั้งหมด

บรรทัดที่ 16 แสดงข้อความในตัวแปร mystr2 เป็นตัวอักษรพิมพ์ใหญ่ทั้งหมด

บทที่ 11 โครงสร้าง (Structure)

11.1 ลักษณะข้อมูลชนิดโครงสร้าง (Structure)

การเก็บข้อมูลเราอาจจำเป็นต้องใช้ข้อมูลที่มีหลายรูปแบบ และเป็นกลุ่มข้อมูลที่มีส่วนร่วมกัน เช่น รายละเอียดของนักศึกษาอาจจะมีข้อมูลรหัสนักศึกษา ชื่อ นามสกุล สาขาวิชา อายุ และเกรดเฉลี่ยของนักศึกษา ซึ่งเราจะต้องสร้างตัวแปรขึ้นมาใช้ในการเก็บข้อมูลของนักศึกษา 6 ตัวแปรด้วยกัน ทั้งที่จริงแล้วข้อมูลทั้งหมด มีส่วนที่เกี่ยวข้องกัน คือเป็นของนักศึกษาคนเดียวกัน หากมีนักศึกษาหลายคน การสร้างตัวแปรและใช้งานตัวแปรอาจจะเกิดความสับสนขึ้นได้ ดังนั้นเราจึงควรนำโครงสร้างข้อมูลมาช่วยในการจัดการกับข้อมูลเหล่านี้

โครงสร้าง (structure) เป็นการกำหนดชนิดข้อมูลใหม่ขึ้นมาใช้งาน โดยชนิดข้อมูลนี้เกิดจากการรวมกันของชนิดข้อมูลพื้นฐานที่อาจจะเป็นชนิดเดียวกัน หรือต่างชนิดกันก็ได้ เพื่อให้เหมาะสมกับข้อมูลที่ต้องการเก็บ

<u>ตัวอย่าง</u> : โครงสร้างข้อมูลนักศึกษาให้ชื่อว่า “student” ประกอบด้วย

รหัสนักศึกษาเป็นชนิด char
ชื่อเป็นชนิด char
นามสกุลเป็นชนิด char
สาขาวิชาเป็นชนิด char
อายุเป็นชนิด int
เกรดเฉลี่ยเป็น float

11.2 การประกาศข้อมูลชนิดโครงสร้าง

การประกาศตัวแปรชนิดโครงสร้างมี 2 รูปแบบดังนี้

<u>รูปแบบที่ 1</u>

```
struct structure_name {
    type field_name_1;
    type field_name_2;
    ...
    type field_name_n;
} variable_name;
```

struct	คือ	ตัวแปรชนิดโครงสร้าง (Structure)
structure_name	คือ	ชื่อของโครงสร้าง
variable_name	คือ	ชื่อของตัวแปรข้อมูลชนิดโครงสร้าง
type field_name_1-n	คือ	ตัวแปรที่เกี่ยวข้องกับตัวแปรข้อมูลชนิดโครงสร้างที่ต้องการเก็บ

ตัวอย่างเช่น

```
struct student {
        char    studentID[13];
        char    name[20];
        char    surname[20];
        char    department[30];
        int     age;
        float   gpa;
} std;
```

สมาชิกข้อมูลภายในมี 6 ฟิลด์ (6 fields)

<u>รูปแบบที่ 2</u>

```
struct structure_name {
    type field_name_1;
    type field_name_2;
    .........
    type field_name_n;
};
struct structure_name variable_name;
```

struct	คือ	ตัวแปรชนิดโครงสร้าง (Structure)
structure_name	คือ	ชื่อของโครงสร้าง
variable_name	คือ	ชื่อของตัวแปรข้อมูลชนิดโครงสร้าง
type field_name_1-n	คือ	ตัวแปรที่เกี่ยวข้องกับตัวแปรข้อมูลชนิดโครงสร้างที่ต้องการเก็บ

ตัวอย่างเช่น

```
struct student {
        char    studentID[13];
        char    name[20];
        char    surname[20];
        char    department[30];
        int     age;
        float   gpa;
} ;
struct student std;
```

(สมาชิกข้อมูลภายในมี 6 ฟิลด์ (6 fields))

<u>ข้อสังเกต</u>

- สมาชิกในโครงสร้างข้อมูล เราเรียกข้อมูลย่อยเหล่านี้ว่า "ฟิลด์ (Field)"
- std เป็นตัวแปรข้อมูลชนิดโครงสร้าง สามารถเก็บข้อมูลนักศึกษาได้ 1 คน ซึ่งก็คือข้อมูล 1 รายการ (1 record)

11.3 การประกาศข้อมูลชนิดโครงสร้างด้วยคีย์เวิร์ด typedef

typedef เป็นคีย์เวิร์ดที่ใช้สำหรับกำหนดชนิดตัวแปรขึ้นมาใหม่

รูปแบบ : typedef ชนิดตัวแปรที่มีอยู่แล้ว ชนิดตัวแปรใหม่; เช่น

typedef int INTERGER;

รูปแบบการใช้ typedef ร่วมกับการนิยามโครงสร้าง :

```
typedef struct{
        type field_name_1;
        type field_name_2;
        .........
        type field_name_n;
} new_definition_variable;
```

ตัวอย่างเช่น

```
typedef struct {
        char     studentID[13];
        char     name[20];
        char     surname[20];
        char     department[30];
        int      age;
        float    gpa;
} student;
```

ในการประกาศตัวแปรก็สามารถใช้ชนิดตัวแปรใหม่ได้เลย เช่น

```
student std1, std2;
```

<u>ข้อสังเกต</u> เมื่อกำหนดตัวแปรหลังจากการกำหนดด้วย typedef แล้วไม่ต้องมีคำว่า struct นำหน้าชนิดข้อมูลอีก

11.4 การกำหนดค่าเริ่มต้นให้กับตัวแปรข้อมูลชนิดโครงสร้าง

ในการกำหนดค่าเริ่มต้นสามารถกำหนดค่าทั้งหมดในตัวแปรได้โดยใช้เครื่องหมาย { } คร่อมค่าเริ่มต้นทั้งหมด และใช้เครื่องหมาย , คั่นระหว่างค่าของสมาชิกแต่ละตัว ตัวอย่างเช่น

```
struct employee {
        char     ssn[10];
        char     fname[20];
        char     lname[20];
        char     department[20];
        int      age;
        float    salary;
} ;
struct employee emp = {“ 999887777”,“Alicia”,” Zelaya”,”Software”,45,25000};
```

จากตัวอย่างทำการกำหนดค่าเริ่มต้นให้กับโครงสร้าง employee โดยกำหนดให้สมาชิก

ssn มีค่าเริ่มต้นเท่ากับ “999887777”

fname มีค่าเริ่มต้นเท่ากับ “Alicia”

lname มีค่าเริ่มต้นเท่ากับ “Zelaya”

department มีค่าเริ่มต้นเท่ากับ “Software”

age ค่าเริ่มต้นเท่ากับ 45

salary ค่าเริ่มต้นเท่ากับ 25000

emp

ssn	999887777
fname	Alicia
lname	Zelaya
department	Software
age	45
salary	25000

11.5 การอ้างถึงสมาชิกในตัวแปรข้อมูลชนิดโครงสร้าง

การอ้างถึงสมาชิกในตัวแปรข้อมูลชนิดโครงสร้าง ทำได้โดยบอกชื่อตัวแปรข้อมูลชนิดโครงสร้างตามด้วยจุด และต่อด้วยชื่อสมาชิกของโครงสร้างนั้น ๆ มีรูปแบบดังนี้

```
structure_name.member;
```

ตัวอย่างเช่น

```
struct student {
        char    studentID[13];
        char    name[20];
        char    surname[20];
        char    department[30];
        int     age;
        float   gpa;
};
struct student student1;
student1.name;          //อ้างถึงข้อมูลชื่อของตัวแปร student1
student1.surname;       //อ้างถึงข้อมูลนามสกุลของตัวแปร student1
student1.gpa;           //อ้างถึงข้อมูลเกรดเฉลี่ยของตัวแปร student1
```

<u>ตัวอย่างที่ 1</u> โปรแกรมแสดงการอ้างถึงสมาชิกในตัวแปรข้อมูลชนิดโครงสร้าง

```c
1  #include <stdio.h>
2  #include <string.h>
3  struct employee {
4  char ssn[10];
5  char fname[20];
6  char lname[20];
7  int age;
8  int salary;
9  };
10
11 int main()  {
12     struct employee emp = {"333445555","John","Smith",42,30000};
13
14     printf("\nssn is %s",emp.ssn);
15     printf("\nFirstname is %s",emp.fname);
16     printf("\nLastname is %s",emp.lname);
17     printf("\nAge is %d",emp.age);
18     printf("\nSalary is %d",emp.salary);
19     return 0;
20 }
```

ผลลัพธ์ของโปรแกรม

```
ssn is 333445555
Firstname is John
Lastname is Smith
Age is 42
Salary is 30000
```

รูปที่ 11.1 ตัวอย่างการอ้างถึงสมาชิกในตัวแปรข้อมูลชนิดโครงสร้าง

จากตัวอย่างโปรแกรมในรูปที่ 11.1 สามารถอธิบายการทำงานดังนี้

บรรทัดที่ 1 – 2 ใช้สำหรับเรียกไฟล์ที่โปรแกรมใช้ในการทำงาน

บรรทัดที่ 3 – 9 เป็นการประกาศข้อมูลชนิดโครงสร้าง employee

บรรทัดที่ 12 เป็นกำหนดค่าเริ่มต้นให้กับตัวแปร emp

บรรทัดที่ 14 - 18 เป็นการแสดงผลข้อมูลสมาชิกของตัวแปร emp

11.6 การรับและแสดงผลข้อมูลสมาชิกของโครงสร้าง

ถ้าสมาชิกของโครงสร้างเป็นสายอักขระ (String) เราจะกำหนดตรง ๆ เหมือนสมาชิกที่เป็นชนิดอื่นไม่ได้ วิธีการกำหนดข้อมูลชนิดสายอักขระ ให้ทำการกำหนดผ่านคำสั่ง strcpy หรือรับค่าจากคีย์บอร์ดด้วยคำสั่ง gets หรือ scanf ก็ได้

<u>ตัวอย่างที่ 2</u> โปรแกรมแสดงการรับและแสดงผลข้อมูลสมาชิกของโครงสร้าง

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
struct project {
int pnumber;
char pname[20];
char plocation[20];
char department[20];
};

int main()  {
    char data[20];
    struct project prj;

    printf("Enter project number : ");
    gets(data);
    prj.pnumber = atoi(data);

    printf("Enter project name : ");
    gets(data);
    strcpy(prj.pname,data);

    printf("Enter location : ");
    gets(data);
    strcpy(prj.plocation,data);

    printf("Enter department : ");
    scanf("%s",prj.department);

    printf("\nProject number is %d",prj.pnumber);
    printf("\nProject name is %s",prj.pname);
    printf("\nLocation is %s",prj.plocation);
    printf("\nDepartment is %s",prj.department);
return 0;
}
```

ผลลัพธ์ของโปรแกรม

```
Enter project number : 62
Enter project name : DatabaseSystem
Enter location : Birmingham
Enter department : Software

Project number is 62
Project name is DatabaseSystem
Location is Birmingham
Department is Software
```

รูปที่ 11.2 ตัวอย่างการรับและแสดงผลข้อมูลสมาชิกของโครงสร้าง

จากตัวอย่างโปรแกรมในรูปที่ 11.2 สามารถอธิบายการทำงานดังนี้

บรรทัดที่ 1 – 3 ใช้สำหรับเรียกไฟล์ที่โปรแกรมใช้ในการทำงาน

บรรทัดที่ 4 – 9 เป็นการประกาศข้อมูลชนิดโครงสร้าง project

บรรทัดที่ 12 – 13 เป็นการประกาศตัวแปร data เป็นชนิด char[20] และ prj เป็นชื่อของตัวแปรข้อมูลชนิดโครงสร้าง project

บรรทัดที่ 15 – 33 เป็นการรับค่าและแสดงผลข้อมูลสมาชิกของตัวแปร prj

11.7 การคัดลอกค่าของตัวแปรข้อมูลชนิดโครงสร้าง

การคัดลอก (Copy) ค่าของตัวแปรข้อมูลชนิดโครงสร้างชนิดเดียวกัน สามารถทำได้โดยใช้เครื่องหมายเท่ากับ (=) เช่น

```
struct student {
        char    studentID[13];
        char    name[20];
        char    surname[20];
        char    department[30];
        int     age;
        float   gpa;
};
struct student std1, std2;
```

แทนที่จะกำหนดค่าสมาชิกทีละตัว

```
std2.name = std1.name;
std2.surname = std1.surname;
std2.department = std1.department;
```

สามารถกำหนดค่าเป็นตัวแปรข้อมูลชนิดโครงสร้างได้เลยเช่น std2 = std1;

<u>ข้อสังเกต</u>

- ในการเปรียบเทียบค่าของตัวแปรข้อมูลชนิดโครงสร้างนั้น ให้เปรียบเทียบค่าของสมาชิกแต่ละตัว นำตัวแปรข้อมูลชนิดโครงสร้างมาเปรียบเทียบกันโดยตรงไม่ได้ เช่น หากมีการกำหนดตัวแปรชนิด struct student ชื่อว่า std1 และ std2 จะนำ std1 และ std2 มาเปรียบเทียบกันโดยตรงไม่ได้ เช่น การใช้คำสั่ง

```
if(std1 == std2)
    printf("std1 is the same data as std2");
```

ดังนั้นการเปรียบเทียบสามารถกระทำได้โดย

```
if(stcmp(std1.name, std2.name) == 0 && stcmp(std1.surname, std2.surname) == 0
        && std1.age == std2.age))
    printf("std1 is the same data as std2");
```

ตัวอย่างที่ 3 โปรแกรมแสดงการคัดลอกค่าของตัวแปรข้อมูลชนิดโครงสร้างชนิดเดียวกัน

```c
#include <stdio.h>
#include <string.h>
struct employee {
char ssn[10];
char name[50];
int age;
char sex;
int salary;
};

int main()  {
    struct employee emp1 = {"555555500","Tom Brand",47,'M',81000};
    struct employee emp2 = {"999887777","Kim Grace",37,'F',25000};

    printf("Before copy");
    printf("\nssn = %s",emp1.ssn);
    printf("   Name = %s",emp1.name);
    printf("   Age = %d",emp1.age);
    printf("   Sex = %c",emp1.sex);
    printf("   Salary = %d",emp1.salary);
    printf("\nssn = %s",emp2.ssn);
    printf("   Name = %s",emp2.name);
    printf("   Age = %d",emp2.age);
    printf("   Sex = %c",emp2.sex);
    printf("   Salary = %d",emp2.salary);

    emp1 = emp2;
    printf("\n\nAfter copy");
    printf("\nssn = %s",emp1.ssn);
    printf("   Name = %s",emp1.name);
    printf("   Age = %d",emp1.age);
    printf("   Sex = %c",emp1.sex);
    printf("   Salary = %d",emp1.salary);
    printf("\nssn = %s",emp2.ssn);
    printf("   Name = %s",emp2.name);
    printf("   Age = %d",emp2.age);
    printf("   Sex = %c",emp2.sex);
    printf("   Salary = %d",emp2.salary);
    return 0;
}
```

ผลลัพธ์ของโปรแกรม

```
Before copy
ssn = 555555500   Name = Tom Brand   Age = 47   Sex = M   Salary = 81000
ssn = 999887777   Name = Kim Grace   Age = 37   Sex = F   Salary = 25000

After copy
ssn = 999887777   Name = Kim Grace   Age = 37   Sex = F   Salary = 25000
ssn = 999887777   Name = Kim Grace   Age = 37   Sex = F   Salary = 25000
```

รูปที่ 11.3 ตัวอย่างการคัดลอกค่าของตัวแปรข้อมูลชนิดโครงสร้างชนิดเดียวกัน

จากตัวอย่างโปรแกรมในรูปที่ 11.3 สามารถอธิบายการทำงานดังนี้

บรรทัดที่ 1 – 2 ใช้สำหรับเรียกไฟล์ที่โปรแกรมใช้ในการทำงาน

บรรทัดที่ 3 – 9 เป็นการประกาศข้อมูลชนิดโครงสร้าง employee

บรรทัดที่ 12 – 13 เป็นกำหนดค่าเริ่มต้นให้กับตัวแปร emp1 และ emp2

บรรทัดที่ 15 – 25 เป็นการแสดงผลข้อมูลสมาชิกของตัวแปร emp1 และ emp2 ก่อนที่จะมีการคัดลอกค่า

บรรทัดที่ 27 เป็นการคัดลอกค่าของตัวแปรโดยกำหนดให้ emp1 = emp2

บรรทัดที่ 28 – 38 เป็นการแสดงผลข้อมูลสมาชิกของตัวแปร emp1 และ emp2 หลังจากที่มีการคัดลอกค่าแล้ว

11.8 อาร์เรย์ของตัวแปรข้อมูลชนิดโครงสร้าง

ปกติเราจะใช้ข้อมูลชนิดโครงสร้างมากกว่าหนึ่งข้อมูล เช่น โครงสร้างนักศึกษา ประกอบด้วย รหัสนักศึกษา ชื่อ นามสกุล สาขาวิชา เพศ วันเกิด ที่อยู่ และเกรดเฉลี่ย เป็นต้น โดยส่วนมากเราจะมีข้อมูลของนักศึกษาหลายคน ดังนั้นเราสามารถสร้างตัวแปรของนักศึกษาให้เป็นอาร์เรย์ของตัวแปรข้อมูลชนิดโครงสร้าง เพื่อทำการเก็บข้อมูลแบบโครงสร้างไว้เป็นกลุ่มข้อมูลที่เกี่ยวข้องกัน

การประกาศอาร์เรย์ของข้อมูลชนิดโครงสร้างมีรูปแบบดังนี้

```
structure structure_name member[size];
```

struct_name คือ ชื่อของโครงสร้าง

variable_name คือ ชื่อของตัวแปรข้อมูลชนิดโครงสร้าง

ตัวอย่างเช่น

```
struct student {
        char    name[20];
        char    surname[20];
        char    department[30];
        int     age;
        float   gpa;
};
struct student std[5];
```

<u>ข้อสังเกต</u>

- ได้ตัวแปรอาร์เรย์ที่มีสมาชิก 5 ตัว (เพื่อเก็บข้อมูลนักศึกษา 5 คน)
- สมาชิกแต่ละตัวในอาเรย์ คือตัวแปรข้อมูลโครงสร้างชนิด student

การกำหนดค่าเริ่มต้นให้กับอาร์เรย์ของข้อมูลชนิดโครงสร้าง มีรูปแบบดังนี้

```
struct  structure_name member[size] = {
                            {value_list 1},
                            {value_list 2},
                                 ...
                            {value_list n}
};
```

ตัวอย่างเช่น

```
struct employee {
        char    ssn[9];
        char    fname[20];
```

```
        char    lname[20];
        char    department[20];
        int     age;
        float   salary;
    } ;

    struct student stu[3] = { {“123456789”,“John”,”Smith”,”Research”,40,30000},
                              {“555555500”,“John”,”James”,”Software”,47,81000},
                              {“444444400”,“Alex”,”Freed”,”Software”,50,89000}
    };
```

ตัวอย่างที่ 4 โปรแกรมการรับค่าและแสดงค่าอาร์เรย์ของตัวแปรข้อมูลชนิดโครงสร้าง

```
#include <stdio.h>
#include <string.h>
struct student {
char name[20];
char surname[20];
int age;
float gpa;
}std[3];

int main()  {
    int i;
    for(i=0;i<3;i++) {
        printf("Enter name of student %d : ",i+1);
        scanf("%s",std[i].name);
        printf("Enter surname : ");
        scanf("%s",std[i].surname);
        printf("Enter age: ");
        scanf("%d",&std[i].age);
        printf("Enter gpa : ");
        scanf("%f",&std[i].gpa);
    }
```

```
22      printf("===============================================\n");
23      printf("Name                 Age            Gpa\n");
24      printf("===============================================\n");
25        for(i=0;i<3;i++) {
26            printf("%s %s",std[i].name, std[i].surname);
27            printf("%8d%16.2f\n",std[i].age,std[i].gpa);
28        }
29  return 0;
30  }
```

ผลลัพธ์ของโปรแกรม

```
Enter name of student 1 : Benjamin
Enter surname : Bayer
Enter age: 19
Enter gpa : 3.21
Enter name of student 2 : Chris
Enter surname : Davidson
Enter age: 25
Enter gpa : 3.53
Enter name of student 3 : Barbara
Enter surname : Benson
Enter age: 19
Enter gpa : 3.25
============================================
Name                Age            Gpa
============================================
Benjamin Bayer      19             3.21
Chris Davidson      25             3.53
Barbara Benson      19             3.25
```

รูปที่ 11.4 ตัวอย่างการรับค่าและแสดงค่าอาร์เรย์ของตัวแปรข้อมูลชนิดโครงสร้าง

จากตัวอย่างโปรแกรมในรูปที่ 11.4 สามารถอธิบายการทำงานดังนี้

บรรทัดที่ 1 – 2 ใช้สำหรับเรียกไฟล์ที่โปรแกรมใช้ในการทำงาน

บรรทัดที่ 3 – 8 เป็นการประกาศข้อมูลชนิดโครงสร้าง student และตัวแปร std[3] เป็นตัวแปรอาร์เรย์ของโครงสร้าง

บรรทัดที่ 11 เป็นการประกาศตัวแปร i เป็นชนิด int

บรรทัดที่ 12 – 21 เป็นการวนลูปรับค่าอาร์เรย์ของตัวแปร std ทั้งหมด 3 รอบ (รับข้อมูลทั้งหมด 3 คน)

บรรทัดที่ 22 – 28 เป็นการแสดงผลข้อมูลสมาชิกของตัวแปร std ทั้งหมด

11.9 ฟังก์ชันกับโครงสร้าง

ตัวแปรข้อมูลชนิดโครงสร้างกับฟังก์ชันสามารถทำได้หลายลักษณะ ทั้งการให้ฟังก์ชันคืนค่าเป็นตัวแปรข้อมูลชนิดโครงสร้าง การส่งอาร์กิวเมนท์ให้ฟังก์ชันเป็นตัวแปรข้อมูลชนิดโครงสร้าง

ถ้าในโปรแกรมมีหลายฟังก์ชัน การนิยามโครงสร้างและการกำหนดชนิดตัวแปรใหม่ (typedef) ให้นำมาไว้นอกฟังก์ชัน main

ตัวอย่างที่ 5 โปรแกรมแสดงฟังก์ชันที่มีการรับค่าเข้าเป็นอาร์เรย์ของโครงสร้าง

```c
#include <stdio.h>
#include <string.h>
typedef struct {
char ssn[10];
char fname[20];
char lname[20];
float salary;
}employee;

void display(employee emps[],int n);
int main() {
  employee emp[3];
  int i, n = 3;
  for (i=0; i<3; i++ )  {
    printf("Enter ssn of employee %d: ",i+1);
    scanf("%s",emp[i].ssn);
    printf("Enter firstname : ");
    scanf("%s",emp[i].fname);
    printf("Enter lastname : ");
    scanf("%s",emp[i].lname);
    printf("Enter salary : ");
    scanf("%f",&emp[i].salary );
  }
  display(emp, n);
  return 0;
}
```

```
27
28  void display (employee emps[], int n)
29  {
30     int i;
31     printf("-----------------------------------\n");
32     printf("Ssn              Name             Salary\n");
33     printf("-----------------------------------\n");
34     for( i=0; i<3; i++) {
35        printf("%s %11s %s",emps[i].ssn,emps[i].fname, emps[i].lname);
36              printf("%10.0f\n",emps[i].salary);
37     }
38  }
```

ผลลัพธ์ของโปรแกรม

```
Enter ssn of employee 1: 987654321
Enter firstname : Bonnie
Enter lastname : Bays
Enter salary : 77000
Enter ssn of employee 2: 987987987
Enter firstname : Arnold
Enter lastname : Head
Enter salary : 81000
Enter ssn of employee 3: 999888777
Enter firstname : Justin
Enter lastname : Mark
Enter salary : 44500
-------------------------------------
Ssn              Name            Salary
-------------------------------------
987654321        Bonnie Bays     77000
987987987        Arnold Head     81000
999888777        Justin Mark     44500
```

รูปที่ 11.5 ตัวอย่างฟังก์ชันที่มีการรับค่าเข้าเป็นอาร์เรย์ของโครงสร้าง

จากตัวอย่างโปรแกรมในรูปที่ 11.5 สามารถอธิบายการทำงานดังนี้

บรรทัดที่ 1 – 2 ใช้สำหรับเรียกไฟล์ที่โปรแกรมใช้ในการทำงาน

บรรทัดที่ 3 – 8 เป็นการประกาศข้อมูลชนิดโครงสร้าง employee

บรรทัดที่ 10 เป็น Function Prototype

บรรทัดที่ 12 เป็นการประกาศตัวแปร emp[3] เป็นตัวแปรอาร์เรย์ของโครงสร้าง

บรรทัดที่ 13 เป็นการประกาศตัวแปร i และ n เป็นชนิด int โดยที่กำหนดให้ n = 3

บรรทัดที่ 14 – 23 เป็นการวนลูปรับค่าอาร์เรย์ของตัวแปร emp ทั้งหมด 3 รอบ (รับข้อมูลทั้งหมด 3 คน)

บรรทัดที่ 24 เป็นการเรียกใช้ฟังก์ชัน display โดยการส่งค่า emp และ n เข้าไปทำงานในฟังก์ชัน

บรรทัดที่ 28 – 37 เป็นฟังก์ชัน display โดยรับค่าที่ส่งมาจากฟังก์ชันหลักมาทำการแสดงค่าอาร์เรย์ของโครงสร้าง

11.10 บทสรุป

โครงสร้าง (structure) เป็นการกำหนดชนิดข้อมูลใหม่ขึ้นมาใช้งาน โดยชนิดข้อมูลนี้เกิดจากการรวมกันของชนิดข้อมูลพื้นฐานที่อาจจะเป็นชนิดเดียวกัน หรือต่างชนิดกันก็ได้

กรณีที่เป็นอาร์เรย์สามารถใช้เลขดัชนี (Index) ในการอ้างอิงสมาชิกแต่ละตัวในอาร์เรย์ได้ แต่รูปแบบการอ้างถึงสมาชิกในตัวแปรข้อมูลชนิดโครงสร้าง ทำได้โดยบอกชื่อตัวแปรข้อมูลชนิดโครงสร้างตามด้วยจุด และต่อด้วยชื่อสมาชิกของโครงสร้าง

การทำงานร่วมกับโครงสร้าง สามารถนำอาร์เรย์ และฟังก์ชัน มาทำงานร่วมได้

บทที่ 12
ไฟล์ (File)

12.1 บทนำสู่ไฟล์

ไฟล์ หรือ แฟ้มข้อมูล นั้นหมายถึง ข้อมูลสาระสนเทศหรือข้อมูลที่เก็บไว้ในรูปแบบดิจิตอล ซึ่งก่อนมีการสร้างคอมพิวเตอร์ เดิมจะถูกเก็บไว้ในตู้แฟ้มเอกสารของสำนักงานต่างๆ แต่ด้วยวิวัฒนาการด้านคอมพิวเตอร์ทำให้ไฟล์มีความสามารถบันทึกลงในสื่อที่มีคุณสมบัติที่สามารถเก็บลักษณะต่างๆ และ สามารถเก็บข้อมูลได้แบบถาวร ข้อมูลก็จะยังคงอยู่ไม่สูญหาย ซึ่งมีความแตกต่างจากหน่วยความจำ เพราะในหน่วยความจำนั้น ถ้าใช้เก็บไฟล์ข้อมูล การเก็บจะเป็นการเก็บแบบชั่วคราว เมื่อปิดเครื่องคอมพิวเตอร์ ข้อมูลที่ถูกเก็บอยู่ในหน่วยความจำจะเกิดการสูญหาย ดังนั้นการสร้างไฟล์ข้อมูลจึงสามารถบันทึก ลงในอุปกรณ์ที่เรียกว่า (Secondary Storage) อย่างเช่น Hard drive , Memory Card หรือสื่อประเภทอื่นๆ ที่นิยมใช้สำรองข้อมูล ในการเรียกชื่อของไฟล์ นั้นมีการเรียกชื่อ หลากหลายชื่อ ตัวอย่างเช่น แฟ้มข้อมูล, แฟ้มอิเล็กทรอนิกส์, แฟ้มคอมพิวเตอร์, แฟ้มดิจิทัล, ไฟล์ข้อมูล, ไฟล์อิเล็กทรอนิกส์, ไฟล์คอมพิวเตอร์, คอมพิวเตอร์ไฟล์, เอกสารอิเล็กทรอนิกส์ ฯลฯ ลักษณะของไฟล์ อาจจะเป็นโปรแกรม รูปภาพ หรือ ข้อมูลในรูปแบบอื่นๆ ได้ โดยทั่วไปแล้ว ไฟล์ถูกแบ่งออกเป็น 2 ประเภทใหญ่ๆ คือ Text files และ Binary files

12.2 ประเภทของไฟล์

ไฟล์จะแบ่งออกเป็น 2 ประเภท ดังนี้คือ Text files และ Binary files

12.2.1 Text files เป็นไฟล์ที่เก็บข้อมูลในรูปแบบของตัวอักษร โดยไฟล์แบบดังกล่าว นั้นจึงไม่สามารถที่จะเก็บข้อมูลประเภทค่าตัวเลขจำนวนเต็ม หรือ ตัวเลขจุดทศนิยม หรือในลักษณะข้อมูลที่เป็นแบบโครงสร้าง โดยการเก็บ ข้อมูลจะถูกแปลงเป็นเลขฐาน 2 และเก็บในรูปแบบตามรหัส ASCII Code ตัวอย่างไฟล์ประเภทนี้ สามารถเปิดใช้งานในลักษณะข้อความ ไม่สามารถ Execute หรือ Run บนระบบปฏิบัติการได้ แต่สามารถเมื่อเปิดใช้งานบน โปรแกรม อย่างเช่น Text Editor ต่างๆ สามารถอ่านเข้าใจได้ โดยปกติจะมีลักษณะนามสกุลไฟล์ .txt .log หรือ อื่นๆที่เป็น Text files

12.2.2 Binary files เป็นไฟล์ที่เก็บข้อมูลในรูปแบบเฉพาะของคอมพิวเตอร์ ซึ่งข้อมูลที่เก็บลงในลักษณะรหัสไบนารี โดยข้อมูลในไฟล์อาจเป็นเลขจำนวนเต็ม เลขจำนวนจริง ตัวอักษร อาร์เรย์ รูปภาพ เสียง หรือสตรัคเจอร์ก็ได้ โดยวิธีการจัดเก็บนั้นมีกระบวนการใช้ข้อมูลรูปแบบเลขฐาน 2 สำหรับไฟล์

รูปแบบนี้ สามารถ Execute หรือ Run บนระบบปฏิบัติการได้ แต่บางไฟล์ ก็เป็นไฟล์ที่เก็บคุณสมบัติพิเศษต่างๆที่เหนือกว่า ไฟล์ แบบ Text files สำหรับไฟล์ในรูปแบบนี้มีนามสกุลไฟล์ที่หลากหลายตามการใช้งาน ตัวอย่างเช่น .exe .com .dat ฯลฯ

12.3 การทำงานกับไฟล์ตัวอักษร(Text file)

สำหรับการทำงานเกี่ยวกับไฟล์นั้น ในเบื้องต้นกระบวนการสามารถอ่านหรือเขียนข้อมูลลงในไฟล์ได้ เมื่อเกิดการสั่งให้อ่านข้อมูลจากไฟล์ กระบวนการที่เกิดขึ้นคือข้อมูลก็จะถูกส่งจากไฟล์มายังหน่วยความจำแรม เพื่อใช้สำหรับแสดงผลข้อมูลที่อยู่ในไฟล์นั้นๆ และ เมื่อเกิดการสั่งให้เขียนข้อมูลลงไฟล์ข้อมูลก็จะถูกส่งจากหน่วยความจำไปยังไฟล์เพื่อทำการบันทึกข้อมูลลงในไฟล์ ซึ่งหน่วยความจำนี้เรียกว่า “บัฟเฟอร์ (buffer)” โดยที่ บัฟเฟอร์ (buffer) คือพื้นที่หน่วยความจำที่ใช้สำหรับเก็บข้อมูลชั่วคราวเพื่อรอดำเนินการกระทำการต่างๆ กับไฟล์ อย่างเช่น อ่าน เขียน ค้นหา แก้ไข หรือ ลบ ได้ โดยแสดงผลการจำลองการทำงานของไฟล์ดังรูปที่ 12.1

อ่าน : ใช้บัฟเฟอร์สำหรับเก็บข้อมูลที่อ่านได้จากไฟล์ เพื่อนำมาแสดงผล

อ่านไฟล์ → บัฟเฟอร์ → แสดงผลข้อมูลออกทางหน้าจอ

เขียน **:** ใช้บัฟเฟอร์สำหรับเก็บข้อมูลที่ต้องการเขียนลงไฟล์ไว้ชั่วคราว ก่อนส่งข้อมูลไปเขียนเก็บลงในไฟล์

ข้อมูลที่จะเก็บลงไฟล์ → บัฟเฟอร์ → เขียนไฟล์

รูปที่ 12.1 แบบจำลองการทำงานของบัฟเฟอร์ของคอมพิวเตอร์

12.3.1 การทำงานร่วมกับไฟล์ตัวอักษร (Text File)

การเตรียมความพร้อมก่อนใช้งาน Text File จำเป็นต้องใช้ฟังก์ชันที่เกี่ยวกับไฟล์โดยเรียกใช้ผ่านไลบรารีไฟล์ stdio.h สำหรับการทำงานกับไฟล์จะต้องทำผ่านตัวอ้างอิงไฟล์ ซึ่งตัวอ้างอิงไฟล์สามารถกำหนดขึ้นได้โดยใช้ชนิดข้อมูลลักษณะ ไฟล์ และ ตัวชี้ (Pointer) ในการกำหนด มีรูปแบบดังต่อไปนี้

FILE *ชื่อที่กำหนดขึ้นเพื่อใช้อ้างอิงไฟล์

12.3.2 การเปิดและปิดไฟล์แบบตัวอักษร(Text File) และ ไฟล์แบบไบนารี (Binary File)

เป็นกระบวนการสำหรับการใช้งาน Text File อย่างถูกต้อง โดยการเปิดเพื่อกำหนดหน้าที่ของไฟล์ที่เปิดว่ามีจุดประสงเพื่อ อ่าน เขียน เพิ่มเติม หรือแก้ไข ส่วนการปิดการใช้งานเพื่อเป้นการจัดการไฟล์ให้มีการเก็บข้อมูล และไม่เกิดความเสียหายกับไฟล์ข้อมูลได้

การเปิดไฟล์ นั้นมีความจำเป็นต้องเรียกใช้งาน ฟังก์ชันในการเปิดไฟล์ คือ ฟังก์ชัน fopen โดยฟังก์ชันนี้ต้องการพารามิเตอร์ 2 ตัว ตัวแรกคือการระบุชื่อไฟล์ที่ต้องการเปิด ซึ่งอาจจะรวมที่อยู่ของไฟล์ (Path) มีรูปแบบคำสั่งที่ใช้ในการเปิดไฟล์แบบ Text File ดังต่อไปนี้คือ

```
FILE fopen("ชื่อไฟล์ที่ต้องการเปิดใช้งาน" , "โหมดการทำงาน");
```

เมื่อผู้เขียนโปรแกรมสั่งเปิดไฟล์ นั่นหมายถึง การเปิดไฟล์ขึ้น เพื่อดำเนินการใดๆ กับไฟล์ ตามค่าพารามิเตอร์ที่ระบุไว้ในส่วนของ mode ซึ่งโหมดการทำงานของ Text File แบ่งออกเป็น 6 ประเภทดังนี้

ตารางที่ 12.1 โหมดการทำงานของการเปิดไฟล์ และ

Mode	ผลการดำเนินการ
r	**เปิดไฟล์เพื่ออ่านข้อมูล** • ถ้ามีไฟล์ชื่อที่กำหนด โปรแกรมกำหนดตำแหน่งไปยังจุดเริ่มต้นของไฟล์ เพื่อเริ่มอ่านข้อมูลจากไฟล์ • กรณีที่ไม่พบไฟล์ที่กำหนดโปรแกรมจะเกิด Error ขึ้นในตัวโปรแกรม
w	**เปิดไฟล์เพื่อเขียนข้อมูล** • ถ้ามีไฟล์ชื่อที่กำหนด โปรแกรมจะเขียนข้อมูลทับข้อมูลเดิมที่มีอยู่แล้วในไฟล์ • ถ้ายังไม่มีไฟล์ที่กำหนด โปรแกรมจะสร้างไฟล์ขึ้นมาใหม่
a	**เปิดไฟล์เพื่อเขียนข้อมูลต่อท้ายข้อมูลเดิมในไฟล์** • ถ้ามีไฟล์อยู่แล้ว โปรแกรมจะกำหนดตำแหน่งไปที่ตำแหน่งท้ายสุดของไฟล์เพื่อเริ่มต้นเขียนข้อมูลต่อท้ายข้อมูลเดิมในไฟล์
r+	เปิดไฟล์เพื่ออ่านหรือเขียนข้อมูลทับไฟล์ที่กำหนด

Mode	ผลการดำเนินการ
w+	เปิดไฟล์เพื่ออ่านและเขียนและลบข้อมูลของไฟล์ที่กำหนดในขณะเริ่มทำงาน
a+	เปิดไฟล์เพื่ออ่านและเขียนต่อท้ายไฟล์ที่กำหนด

ตัวอย่าง การเปิดไฟล์โดยโปรแกรมภาษาซี

fopen(“test.txt”,“r”);

หรือ

fopen(“D:\files\test.txt”,“r”);

หมายถึง เปิดไฟล์ชื่อ test.txt จากไดเร็คทอรีปัจจุบันที่ใช้งานอยู่ หรือ ตัวอย่างถัดไป เปิดไฟล์ชื่อ test.txt จากโฟลเดอร์ (Folder) ที่มีชื่อว่า files เพื่อ การอ่านข้อมูล

ข้อควรระวัง สำหรับการใช้งานฟังก์ชัน fopen ควรระมัดระวังในเรื่องของที่อยู่หรือ Folder ที่เก็บไฟล์ เพราะถ้าระบุส่วนของ “ชื่อไฟล์ที่ต้องการเปิดใช้งาน” โดยกำหนด Folder ผิดแล้ว ก็จะทำให้เปิดไฟล์ไม่สำเร็จ อาจเกิดข้อผิดพลาดได้ และ ฟังก์ชัน fopen() ส่งค่ากลับ file pointer (fp) เท่ากับ NULL เมื่อมีความผิดพลาดในการเปิดไฟล์หรือไม่สามารถเปิดไฟล์ได้ ในกรณีที่สามารถเปิดแฟ้มข้อมูลได้ โดยฟังก์ชันส่งค่ากลับ file pointer (fp) ไม่เท่ากับ NULL

การปิดไฟล์

รูปแบบคำสั่งสำหรับการปิดไฟล์แบบ Text files มีรูปแบบดังต่อไปนี้คือ

fclose(ชื่อที่กำหนดขึ้นเพื่อใช้อ้างอิงไฟล์)

fclose มีหน้าที่ทำให้ข้อมูลใดๆ ในบัฟเฟอร์ที่ยังไม่ถูกเขียนลงในไฟล์ที่อยู่บนฮาร์ดดิสก์ หรือ แหล่งเก็บข้อมูลถาวร แล้วทำการปิดไฟล์ โดยที่ฟังก์ชัน fclose จะคืนค่า 0 แต่ถ้ามีข้อผิดพลาดเกิดขึ้นจะคืนค่า EOF ด้วยเหตุนี้จึงต้องมีการตรวจสอบการคืนค่าเพราะว่า ในการทำงานกับไฟล์หากทำงานไม่สำเร็จ จะได้ออกแบบโปรแกรมให้แจ้งเตือนผู้ใช้งานว่าการสั่งงานไม่สมบูรณ์

ตัวอย่าง การเปิด และ ปิดไฟล์ โดยโปรแกรมภาษาซี

```
1  #include<stdio.h>
2  main()
3  {
4  FILE *fp;
5  fp = fopen("c:\test.txt" , "r");
6  if (!fp) {
7          printf("Cannot Open test.txt\n");
8      }
9  fclose(fp);
10 }
```

รูปที่ 12.2 ตัวอย่างการเปิด และ ปิดไฟล์ โดยโปรแกรมภาษาซี

จากตัวอย่างโปรแกรมในรูปที่ 12.2 สามารถอธิบายการทำงานดังนี้

บรรทัดที่ 4 สร้างตัวชี้ fp เป็นข้อมูลแบบไฟล์

บรรทัดที่ 5 เป็นการใช้คำสั่ง fopen() เปิดไฟล์ ชื่อ test.txt จากไดรว์ C เพื่อการอ่านอย่างเดียว

บรรทัดที่ 6 – 8 เป็นการตรวจสอบเงื่อนไข กรณีที่ไม่มีมีไฟล์ test.txt อยู่ในไดรว์ C หรือหากเกิดข้อผิดพลาดฟังก์ชัน fopen() ส่งค่ากลับเป็น NULL กลับตัวแปร fp โดยโปรแกรมจะพิมพ์ข้อความ “Cannot Open test.txt” ปรากฏทางหน้าจอคอมพิวเตอร์

บรรทัดที่ 9 เป็นการปิดไฟล์พร้อมบันทึกข้อมูลลงหน่วยความจำถาวรโดยฟังก์ชัน fclose()

หมายเหตุ ต้อง fclose() เพื่อทำการปิดไฟล์ทุกครั้งที่ทำงานกับไฟล์เรียบร้อยแล้ว และ ตัวแปร *fp เป็น file pointer คือ ตัวแทนของไฟล์ การทำงานกับไฟล์ต้องทำผ่าน file pointer เท่านั้น

12.3.3 รูปแบบการอ่านข้อมูลจาก Text files ด้วยฟังก์ชัน fscanf()

การอ่านไฟล์โดยใช้ฟังก์ชัน fscanf() มีรูปแบบการอ่านข้อมูลจากไฟล์ดังนี้ สามารถใช้อ่านข้อมูลจากแฟ้มข้อมูลในรูปแบบตัวเลขจำนวนเต็ม, ตัวเลขจุดทศนิยมและข้อความได้

fscanf(ชื่อที่ใช้อ้างอิงไฟล์, รูปแบบของการอ่านค่าจากแฟ้ม, ตัวแปรสำหรับเก็บค่า);

สำหรับฟังก์ชัน fscanf() คล้ายกับฟังก์ชัน scanf() โดยฟังก์ชันทั้งสองใช้ในการรับข้อมูลเหมือนกัน แต่ฟังก์ชัน scanf() จะทำการรับข้อมูลเข้ามาทางคีย์บอร์ด ส่วน ฟังก์ชัน fscanf() จะรับข้อมูลโดยอ่านมาจากไฟล์

12.3.4 รูปแบบการเขียนข้อมูลลง Text files ด้วยฟังก์ชัน fprintf()

การเขียนไฟล์โดยใช้ฟังก์ชัน fprintf() มีรูปแบบการเขียนข้อมูลลงไฟล์ ดังนี้

fscanf(ชื่อที่ใช้อ้างอิงไฟล์, รูปแบบของการเขียนค่าจากแฟ้ม, ตัวแปรสำหรับเก็บค่า);

จากรูปแบบจะเห็นได้ว่า ฟังก์ชัน fprintf() คล้ายคลึงกับฟังก์ชัน printf() ตรงที่ฟังก์ชันทั้งสองใช้ในแสดงผลข้อมูลเหมือนกันแต่ฟังก์ชัน printf() จะแสดงผลข้อมูลออกทางจอภาพส่วนฟังก์ชัน fprintf() จะแสดงผลข้อมูลโดยเขียนข้อมูลลงไฟล์

```
1  #include <stdio.h>
2  main()
3  {
4  char buff[20];
5  int num;
6      FILE *fp;
7      fp = fopen ("test_file.txt", "w+");
8      fprintf(fp, "%s %s %s %d", "This", "is", "number", 12345);
9      fclose(fp);
10     fp = fopen ("test_file.txt", "r");
11     while(fscanf(fp, "%s", buff)!=EOF){
12         printf("%s ", buff );
13     }
14     fclose(fp);
15 }
```

รูปที่ 12.3 การใช้งาน fprintf() และ fscanf()

จากตัวอย่างโปรแกรมในรูปที่ 12.3 สามารถอธิบายส่วนประกอบที่สำคัญการทำงานดังนี้

บรรทัดที่ 4 สร้างตัวแปรอาเรย์เพื่อเก็บค่าที่มาจากไฟล์ในรูปแบบข้อความ

บรรทัดที่ 7 เป็นการใช้คำสั่ง fopen() เปิดไฟล์ C เพื่อการเขียน

บรรทัดที่ 8 เป็นการเขียนข้อมูลลงไฟล์ โดยคำสั่ง fprintf() ด้วยข้อมูลที่กำหนด

บรรทัดที่ 9 เป็นการปิดไฟล์พร้อมบันทึกข้อมูลลงหน่วยความจำถาวรโดยฟังก์ชัน fclose()

บรรทัดที่ 10 เป็นการใช้คำสั่ง fopen() เปิดไฟล์ C เพื่อการอ่านอย่างเดียว

บรรทัดที่ 11 อ่านค่าอักษรใน Text files โดยคำสั่ง fscanf() เก็บในตัวอาเรย์ buff

บรรทัดที่ 12 พิมพ์ข้อความที่เก็บในตัวอาเรย์ buff

บรรทัดที่ 14 ปิดไฟล์โดยฟังก์ชัน fclose()

12.3.5 รูปแบบการลบไฟล์ข้อมูลจาก Text files ด้วยฟังก์ชัน remove()

มีรูปแบบ คือ

remove(ชื่อไฟล์ข้อมูล)

ตัวอย่าง

```
result = remove("test_files.txt");
```

หมายเหตุ ฟังก์ชันนี้ เมื่อลบไฟล์สำเร็จจะให้ค่า 0 หรือ เท็จ กลับมา ถ้าลบไม่สำเร็จ จะให้ค่าอื่นที่ไม่ใช่ 0 ดังนั้นการเขียนโปรแกรมเพื่อตรวจสอบจึงต้องระวัง

12.3.6 รูปแบบการเปลี่ยนชื่อไฟล์ข้อมูลจาก Text files ด้วยฟังก์ชัน rename()

มีรูปแบบดังนี้ คือ

rename(old name ,new name)

ตัวอย่าง

```
result = rename("test_files.txt ","new_files.txt ");
```

หมายเหตุ ฟังก์ชันนี้ เมื่อเปลี่ยนชื่อไฟล์สำเร็จจะให้ค่า 0 หรือ เท็จ กลับมา ถ้าเปลี่ยนชื่อไม่สำเร็จ จะให้ค่าอื่นที่ไม่ใช่ 0 ดังนั้นการเขียนโปรแกรม เพื่อตรวจสอบจึงต้องระวัง

จะเห็นได้ว่า มีฟังก์ชันที่มีการทำงานเกี่ยวข้องกับไฟล์หลายฟังก์ชัน โดยที่กล่าวมาข้างต้น เป็นฟังก์ชันที่มีความจำเป็นในการใช้งาน ยังมีฟังก์ชันที่ยังไม่ได้กล่าวถึง แสดงดังตารางที่ 12.2

ตารางที่ 12.2 โหมดการทำงานของการเปิดไฟล์ และ

ฟังก์ชัน	การทำงาน
fputc()	ใช้สำหรับเขียนตัวอักษรลงไปบนไฟล์
fputs()	ใช้สำหรับเขียนข้อความที่เป็น string ลงไปบนไฟล์
fgetc()	ใช้สำหรับอ่านข้อมูลหนึ่งตัวอักษรจากไฟล์
fgets()	ใช้สำหรับการอ่านข้อความเป็นบรรทัดจากไฟล์

12.4 การทำงานกับไบนารีไฟล์

สำหรับโหมดการทำงานของไบนารีไฟล์แตกต่างจากโหมดการทำงานของเท็กซ์ไฟล์ คือ ในแต่ละโหมดการทำงานของไบนารีไฟล์จะระบุตัวอักษร ‘b’ ต่อท้ายไว้ที่โหมดนั้นๆ ด้วย เพื่อบ่งบอกว่าเป็นโหมดการทำงานของไบนารีไฟล์ ดังในตารางที่ 12.3

ตารางที่ 12.3 โหมดการทำงานของการเปิดไฟล์แบบไบนารี

Mode	ผลการดำเนินการ
rb	เปิดไฟล์แบบไบนารีเพื่ออ่านข้อมูลจากตำแหน่งเริ่มต้นของไฟล์ ถ้าไม่มีไฟล์อยู่จะเกิด Error ขึ้น
wb	เปิดไฟล์แบบไบนารีเพื่อเขียนข้อมูล โดยถ้ามีไฟล์อยู่แล้วจะเขียนข้อมูลทับไฟล์เดิม แต่ถ้ายังไม่มีไฟล์จะสร้างไฟล์ขึ้นใหม่
ab	เปิดไฟล์แบบไบนารีเพื่อเขียนข้อมูล โดยถ้ามีไฟล์อยู่แล้วจะเขียนข้อมูลต่อท้ายไฟล์ข้อมูลที่มีอยู่แล้วในไฟล์ แต่ถ้ายังไม่มีไฟล์จะสร้างไฟล์ขึ้นใหม่
rb+	เปิดไฟล์เก่าเพื่ออ่านหรือเขียนข้อมูลทับข้อมูลเดิม
wb+	เปิดไฟล์เก่าเพื่ออ่านหรือเขียนข้อมูลทับข้อมูลเดิม หรือสร้างไฟล์ใหม่
ab+	เปิดไฟล์เก่าเพื่ออ่านหรือเขียนข้อมูลต่อท้ายข้อมูลเดิม หรือสร้างไฟล์ใหม่

ฟังก์ชันที่มีการใช้งานกับไฟล์ไบนารี มี 2 ฟังก์ชันดังต่อไปนี้

12.4.1 การอ่านข้อมูลจากไบนารีไฟล์ด้วยฟังก์ชัน fread()

มีรูปแบบดังนี้คือ

fread(*ptr,ขนาดของข้อมูล,จำนวนข้อมูลที่ต้องการอ่าน,ชื่อที่กำหนดขึ้นเพื่อใช้อ้างอิงไฟล์);

โดยฟังก์ชัน fread() เป็นฟังก์ชันที่ใช้อ่านข้อมูลจากไฟล์ไบนารี โดยแต่ละครั้งสามารถกำหนดขนาด (size) ของข้อมูลที่ต้องการอ่านได้ ตัวอย่างเช่น

```
char a[40];
fread(&a,sizeof(char),5,fp);
```

จากตัวอย่างที่แสดง ฟังก์ชัน fread() เป็นการอ่านข้อมูลในไฟล์มาเก็บไว้ที่ตัวแปร a โดยทำการอ่านข้อมูล 5 ครั้ง (**จำนวนข้อมูลที่ต้องการอ่าน**) ขนาดของข้อมูลที่อ่านจากไฟล์แต่ละครั้งมีขนาด 1 ไบต์ ตามชนิดตัวแปร a ดังนั้นข้อมูลทั้งหมดที่อ่านจากไฟล์ จะมีขนาดเท่ากับ 5 ไบต์

หมายเหตุ ฟังก์ชัน fread( ) จะให้ค่าตัวเลขจำนวนเต็ม = n เมื่อไม่เกิดข้อผิดพลาดในการอ่านข้อมูล และจะให้ค่าเป็นศูนย์ (0 หรือ NULL) เมื่อมีข้อผิดพลาดในการอ่านข้อมูลจากแฟ้มหรือสิ้นสุดไฟล์ (EOF)

12.4.2 การเขียนข้อมูลลงไบนารีไฟล์ด้วยฟังก์ชัน fwrite()

ในการเขียนข้อมูลลงไบนารีไฟล์ด้วยฟังก์ชัน fwrite() นั้นจะมีหลักการทำงานเดียวกันกับการทำงานของฟังก์ชัน fread() ทุกประการ โดย fwrite() มีรูปแบบในการเขียนข้อมูล
ดังนี้คือ

fwrite(*ptr,ขนาดของข้อมูล,จำนวนข้อมูลที่ต้องการเขียน,ชื่อที่กำหนดขึ้นเพื่อใช้อ้างอิง

ฟังก์ชัน fwrite( ) เป็นฟังก์ชันที่ใช้เก็บข้อมูล ลงไฟล์ไบนารีโดยที่การเก็บข้อมูลแต่ละครั้ง สามารถกำหนดขนาดของข้อมูลที่ต้องการบันทึกได้ ตัวอย่างเช่น

```
int  x=120;

fwrite(&x,sizeof(int x),1,fp);
```

จากตัวอย่าง เป็นการบันทึกข้อมูลที่เก็บไว้ในตัวแปร x ลงไฟล์ โดยทำการบันทึก 1 ครั้ง (**จำนวนข้อมูลที่ต้องการเขียน**) ขนาดของข้อมูลที่บันทึกลงแฟ้มในแต่ละครั้งมีขนาด 2 bytes ตามชนิดตัวแปร x

โดยมีโปรแกรมตัวอย่างการใช้งาน ดังรูปที่ 12.4 ซึ่งเป็นการใช้งานกับไฟล์ที่เป็นไบนารีไฟล์ โดยมีการอธิบายการทำงานใต้ภาพประกอบ

หมายเหตุ ฟังก์ชัน fwrite( ) จะให้ค่าเลขจำนวนเต็มเท่ากับ n เมื่อไม่เกิดข้อผิดพลาดในการเขียนข้อมูล และจะให้ค่าน้อยกว่า n เมื่อมีข้อผิดพลาดเกิดขึ้น

```c
1  #include <stdio.h>
2  struct dat
3  {
4      int mm;
5      int dd;
6      int yy;
7  };
8  main()
9  {
10     int i;
11     FILE *b_file;
12     struct dat d;
13     b_file = fopen("date_file", "wb");
14     d.mm=12;
15     d.dd=31;
16     d.yy=22;
17     fwrite(&d, sizeof(struct dat), 1, b_file);
18     fclose(b_file);
19 
20     b_file = fopen("date_file", "rb");
21         fread(&d, sizeof(struct dat), 1, b_file);
22         printf("Month:%d Day:%d Year:%d\n", d.mm, d.dd, d.yy);
23     fclose(b_file);
24 }
```

รูปที่ 12.4 การใช้งาน fwrite() และ fread()

จากตัวอย่างโปรแกรมในรูปที่ 12.4 สามารถอธิบายส่วนประกอบที่สำคัญการทำงานดังนี้

บรรทัดที่ 2-6 สร้างตัวแปรแบบโครงสร้างเพื่อเก็บ วัน เดือน ปี

บรรทัดที่ 10-12 เป็นการประกาศตัวแปร ที่ใช้งาน รวมถึงตัวแปรไฟล์ และ โครงสร้าง

บรรทัดที่ 13 เป็นการใช้คำสั่ง fopen() เปิดไฟล์ C เพื่อการเขียน

บรรทัดที่ 14-16 เป็นการให้ค่าตัวแปรแบบโครงสร้าง

บรรทัดที่ 17 เป็นการใช้คำสั่ง fwrite() เขียนไฟล์ไบนารี เท่ากับขนาด 6 ไบต์ เท่ากับขนาดโครงสร้าง

บรรทัดที่ 21 เป็นการใช้คำสั่ง fread() เพื่ออ่านข้อมูลในไฟล์

บรรทัดที่ 22 พิมพ์ข้อความที่อ่านจากไฟล์

บรรทัดที่ 23 ปิดไฟล์โดยฟังก์ชัน fclose()

12.5 บทสรุป

ไฟล์ หรือ ชื่อเรียกในแบบอื่นๆ ที่กล่าวในข้างต้น เป็นชื่อที่ใช้ในการอ้างอิงข้อมูลที่ได้จัดเก็บไว้ในหน่วยความจำถาวร อย่างเช่น ฮาร์ดดิสก์ (Hard disk) เพื่อเรียกใช้ข้อมูลที่อยู่ในไฟล์ได้ โดยผู้เขียนโปรแกรมต้องมีการอ้างอิงถึงชื่อไฟล์ที่ได้ทำงานจัดเก็บนั้น โดยไฟล์ แบ่งออกได้เป็น 2 ประเภท

1. ไฟล์ข้อความ (Text file) เป็นไฟล์ที่เก็บข้อความ ไฟล์นี้จะเก็บข้อมูลเป็นรหัส ASCII รูปแบบตัวอักษร และสามารถเปิดดูเห็นข้อความนั้น และเข้าใจความหมายได้

2. ไฟล์ไบนารี (Binary file) ไฟล์ที่ได้ถูกแปลเป็นภาษาเครื่อง หรือเป็นเลขฐานสอง มนุษย์ไม่เข้าใจความหมาย

สำหรับการทำงานร่วมกับไฟล์ สามารถจัดการไฟล์ได้หลากหลายวิธีด้วยกัน ดังต่อไปนี้ การเปิดไฟล์การปิดไฟล์ การอ่านไฟล์ การเขียนไฟล์ การค้นหาตำแหน่งไฟล์ การลบไฟล์ และ การเปลี่ยนชื่อไฟล์

บรรณานุกรม

[1] Chan, T. **Unix System Programming Using C++** , Prentice Hall PTR, 1997, 598 p.

[2] Deitel, H.M. , **C How to Program Third Edtion**, Prentice Hall,2001 , 1253 p.

[3] Frank L.Friedman and Elliot B. Koffman, 2000, **Problem sloving, abstraction, and design using C++**, Addison Wesley Longman, Inc.

[4] Herbert Schildt , **C Power User's Guide,** Osborne McGraw-Hill , 1988.

[5] Les Hancock , Morris Krieger , **The C Primer** , McGraw-Hill Book Company , 1998.

[6] Microsoft Press, **Microsoft Visual C++ Version 6.00**, Microsoft Corporation, 1998.

[7] Peter Ailtken & Bradley Jones,**Teach Yourself C in 21 Days**, Sams Publishing, 1994.

[8] Robert Lafore, **Object-Oriented Programming in Turbo C++**, The Waite Group, 1991.

[9] Thomson, **C++ Programming from Problem Analysis to Program Design** , 2ndEdition,2004

[10] V. Zwass, 2004, **Foundations of information systems**, McGraw-Hill International Editions

[11] Yuda, **Introduction to C++ Programming** , Korona-sha (2005)

[12] มนต์ชัย เทียนทอง, **การเขียนโปรแกรมภาษาซี** , โรงพิมพ์สถาบันเทคโนโลยีพระจอมเกล้าพระนครเหนือ กรุงเทพฯ, 2535.

[13] บุญเลิศ เอี่ยมทัศนา , ยืน ภู่วรวรรณ , สมนึก คีรีโต , **โปรแกรมคอมพิวเตอร์ภาษาซี** กรุงเทพฯ : ซีเอ็ดยูเคชั่น จ ากัด , 2521.

[14] ผศ.ดร. ดวงแก้ว สวามิภักดิ์ , **การโปรแกรมภาษา C** , กรุงเทพฯ : ซีเอ็ดยูเคชั่น , 2521.

[15] ยุทธนา ลีลาศวัฒนกุล, **เริ่มต้นการเขียนโปรแกรมด้วยภาษา C++**, ไทยเจริญการพิมพ์ 2547

[16] อรพิน ประวัติบริสุทธิ์, คู่มือเขียนโปรแกรมด้วยภาษา C, กรุงเทพฯ : โปรวิชั่น, 2556

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

แหล่งที่มา https://commons.wikimedia.org/wiki/File:ASCII-Table-wide.svg